UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LEONARDO AUGUSTO SCHMITZ

Analysis and Acceleration of High Quality Isosurface Contouring

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Prof. Dr. João L. D. Comba
Advisor

Porto Alegre, October 2009

CIP – CATALOGING-IN-PUBLICATION

Schmitz, Leonardo Augusto

Analysis and Acceleration of High Quality Isosurface Contouring / Leonardo Augusto Schmitz. – Porto Alegre: PPGC da UFRGS, 2009.

80 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2009. Advisor: João L. D. Comba.

1. Isosurface extraction. 2. Volumetric visualization. 3. Contouring. 4. Graphics hardware architecture. I. Comba, João L. D.. II. Title.

"...but that's just mapping the algorithm to the GPU!"

— SET OF ANONYMOUS REVIEWERS

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor João Luiz Dihl Comba for his support in these two years of research. His guidance helped me in the improvement of my critical thinking and discussion skills. I also appreciated his advices about priorities, which helped me overcome a chaotic scheduling of tasks. He is always looking for the best interest of his researchers.

I would also like to thank Carlos A. Dietrich, who could be considered an informal co-advisor in the elaboration of this thesis. His large experience in isosurface extraction helped me avoid pitfalls and speed up the pursuit of relevant research contributions. He also contributed with a significant number of the ideas to the thesis.

I want to thank everyone from the UFRGS Computer Graphics Group for their time spent in fruitful discussions and for their help in quickly referencing commands in technologies such as TeX, which was used to the writing of this thesis. Particularly, I would like to thank Andre Spritzer and Vitor Pamplona for additional corrections in English and coherence in the text. A special thanks go for our CG professors Manuel Menezes de Oliveira Neto, Carla Maria Dal Sasso Freitas and Luciana Porcher Nedel. I also want to thank our government sponsors CNPq (process number 136569/2008-0) and CAPES for their financial support.

Finally, I want to thank God and my family. This includes my dad, Paulo Alberto Schmitz, my mom, Aneli Schmitz, my girlfriend, Luciane Ramos and my siblings, Carla and Gustavo. They are an essential part for everyday in my life. They are responsible for the paths I have taken and for my growth as a human being. Without any of them, I definitively would not have finished this thesis.

Thank you very much, everyone.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This work presents an analysis of the GPU implementations of the main isosurface polygonization algorithms. The result of the analysis shows how the GPU may be modified in order to support polygonization of isosurfaces and also how these algorithms are adapted to fit current GPUs. The techniques used in GPU-based versions of Marching Cubes are extended and a polygonization with improved quality is achieved. We propose parallel versions of Dual Contouring and Macet, algorithms which improve accuracy and shape of triangles meshes, respectively. Both GPU approaches extract isosurfaces from large volumetric data in less than one second, improving CPU versions up to two orders of magnitude.

The contributions of this work also include a novel table-driven approach of Dual Contouring (DC) for uniform grids. A table is used for quad topology specification, which aids the implementation and cache efficiency of parallel scenarios. It is suitable for stream expansion with GPU versions of both geometry shader and Histogram Pyramids. In addition, our isosurface feature approximation version of DC is more straightforward than Singular Value and QR Decompositions. The vertex positioning does not require diagonalization of matrices. Instead, it uses a simple trilinear interpolation.

In order to evaluate the efficiency of the techniques presented in this work, we compare our approaches to state-of-the-art Marching Cubes GPU versions. We also present a deep analysis of GPU architectures for isosurface extraction, in which industry profiling tools are used. This analysis shows bottlenecks of the graphics hardware and helps the evaluation of possible solutions for next generation GPUs.

Keywords: Isosurface extraction, Volumetric visualization, Contouring, Graphics hardware architecture.

Análise e Aceleração da Extração de Isosuperfícies com Alta Qualidade

RESUMO

Este trabalho apresenta uma análise dos principais algoritmos de poligonização de isosuperfícies na GPU. O resultado desta análise mostra tanto como a GPU pode ser modificada para oferecer suporte a este tipo de algoritmo quanto como os algoritmos podem ser modificados para se adaptar as características das GPUs atuais. As técnicas usadas em versões de GPU do Marching Cubes são extendidas e uma poligonização com menos artefatos é gerada. São propostas versões paralelas do Dual Contouring e do Macet, algoritmos que melhoram a aproximação e a forma das malhas de triângulos, respectivamente. Ambas técnicas extraem isosuperfícies a partir de grandes volumes de dados em menos de um segundo, superando versões de CPU em até duas ordens de grandeza.

As contribuições desse trabalho incluem uma versão orientada a tabelas do Dual Contouring (DC) para *grids* estruturados. A tabela é utilizada na especificação da topologia dos quadriláteros, que ajuda a implementação e a eficiência de cache em cenários paralelos. A tabela é adequada para a expansão de *streams* na GPU em ambos *geometry shader* e Histogram Pyramids. Além disso, nossa versão de aproximação de características das isosuperfícies é mais simples que a Decomposição de Valores Singulares e também que a Decomposição QR. O posicionamento dos vértices não requer uma diagonalização de matrizes. Ao invés disso, usa-se uma simples interpolação trilinear.

Afim de avaliar a eficiência das técnicas apresentadas neste trabalho, comparamos nossas técnicas com versões do Marching Cubes na GPU do estado da arte. Também incluímos uma análise detalhada da arquitetura de GPU para a extração de isosuperfícies, usando ferramentas de avaliação de desempenho da indústria. Essa análise apresenta os gargalos das placas gráficas na extração de isosuperfícies e ajuda na avaliação de possíveis soluções para as GPUs das próximas gerações.

Palavras-chave: Extração de isosuperfícies, Visualização volumétrica, Contorno, Arquitetura da placa gráfica.

1 INTRODUCTION

Scientific visualization of volume data has been an important focus for many computer graphics researchers. The creation of insightful visual images from volume data aids scientists in the understanding of complex environments, ranging from physics to medicine. Images are more appealing than other sources of information for most people, since vision is our dominant sense and is consequently responsible for a large part of human cognition (BAINES, 2008). Thus, rendering meaningful images can be a helpful tool to the advance of multiple scientific branches.

The acquisition of volume data is common in many distinct areas of science. Physicians extract volume data from patients in magnetic resonance imaging (MRI) and computer tomography (CT) for diagnosis; engineers acquire volume data from cars and airplanes in wind-tunnels to test for aerodynamics; engineers also acquire data from numerical approximations of partial differential equations, such as heat transfer in engines, to test material resistances; meteorologists obtain data about air density and pressure for climate forecasts; *etc*.

The scientific volumes consist of a set of points uniformly or non-uniformly distributed with associated scalar values, functions or groups of functions. These volumes can be viewed as discrete functions bounded in a regular or irregular space. When a volume has uniformly distributed points, it is referred to as a regular volume grid. Otherwise, it is referred to as irregular. Image scanners (MRIs and CTs) usually obtain its point sampled densities on regular grids, while many simulations obtain irregular ones. Current image scanners and simulation techniques may produce very large volumetric images. This massive numerical information implies in two challenges for the extraction of images: (i) how to choose parts of the volume to produce meaningful images; (ii) how to do it efficiently.

The visualization of isosurfaces is a common approach to filter large volumetric datasets. Isosurfaces are implicit surfaces defined by the zero-value of a given function ($f : R^n \rightarrow R$). The choice of one or more isovalues i simplifies the volume into areas of interest. The isovalue is the only parameter in the definition of the isosurface $I(i) = \{x|f(x) - i = 0\}$. Cartographers often use a set of isocontours defined by multiple isolevels (heights) in their topographic maps (2D function, where $n = 2$). Contours are important for understanding elevations of irregular terrains (see Figure 1.1). A set of connected points with the same height represents each contour. The set of these contours in the 2D image contains enough information to give a 3D perspective of the place. Isosurfaces in 3D ($n = 3$) are an extension of this. The set of points with equal isovalue represents a surface.

The representation of these isosurfaces is a challenging problem and much research has been done in the last decades. Some methods extract surfaces without explicit connectivity. These methods may use point primitives such as surfels (CO; HAMANN; JOY,

2003) or may even cast rays through the volume for direct visualization (KRUGER; WESTERMANN, 2003). The first approach is memory and processing efficient. The latter is accurate and produces beautiful isosurfaces. However, both suffer from their lack of topology. If an application (collision detection, for example) requires that a primitive uses boundary information from its neighbors, both representations are ineffective. This is a serious issue if the generated surfaces are used as input for other application. On the other hand, methods that polygonize the isosurface (WUENSCHE, 1997) have explicit topology and do not suffer from these problems.

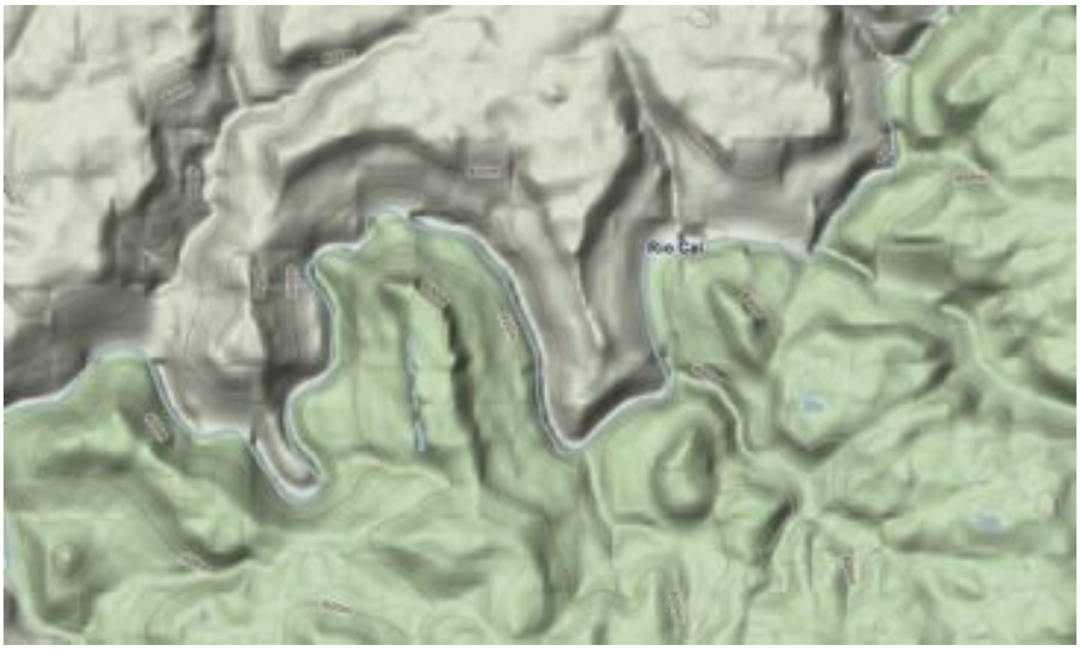


Figure 1.1: Cartography contour lines. Isocontours represent different elevations on the terrain. Each line has one height associated. With this visualization technique, it is much easier to understand the position of peaks and depressions in the terrain. In this particular image, the relief shading helps the visualization even further. Image extracted from Google Maps®.

Lorensen and Cline's Marching Cubes (MC) (LORENSEN; CLINE, 1987) is one of the first and the most successful isosurface polygonizer. In brief, it divides the space into an uniform grid of cubes and generates triangles for the ones trespassed by the isosurface based on a table of predefined cases. The major causes for such acceptance are its simplicity, efficiency and robustness. Unfortunately, the algorithm has some shortcomings. The triangle meshes generated might have degenerate triangles. In addition, the triangle meshes fail to represent sharp edges and corners, due to the assumption of MC that the input function is smooth.

Our first goal in this work is to present high-quality polygonization of isosurfaces. Applications that create these polygonizations should have at least one of two possible advantages over low-quality polygonizations. First, the polygonization generated is closer to the isosurface. Second, the polygonization can be used as input by applications that require non-degenerate polygons. These advantages can be achieved by using variations of the original MC method presented on literature. Many variations need some extra computational effort, which compromise the efficiency of the original algorithm.

Our second goal is to maintain interactive frame-rates in the polygonization of isosurfaces. A single isosurface is just a small part of the information contained in the volume. Although it can be useful to show the local behavior of the information contained in the volume, the global behavior can be only unveiled by showing a huge amount of isosurfaces at the same time (in the same way that it is done in 2D height maps) or by having the ability to change the isolevel interactively. There are three common approaches for the acceleration of the polygonization step. First is to avoid empty space around the isosurfaces, which is explored by one of our solutions. Second is to polygonize the isosurface in a view-dependent way, which is not interesting to us as it does not accelerate the extraction of the whole isosurface. Finally, the third is to use massive parallel hardware such as the Graphics Processor Units (GPUs), which is one of the most important focuses of this work.

The GPU represents a programmable and generic processor (MACEDONIA, 2003) and several algorithms explore its computational power. The GPU is a very powerful tool for almost every parallel application. It distributes tasks for its multiple cores and several algorithms are faster than on the CPU. Even though there is an increasing parallel power on multi-core CPUs, we believe the most promising low-cost hardware today for the polygonization of isosurface is the one on the GPUs. The intrinsic parallel nature of the most common isosurfacing algorithms such as MC makes it amenable for parallel (and thus faster) implementations.

1.1 Isosurface Applications

As mentioned before, isosurfaces are useful for filtering the content of massive volumetric data. There are infinite isovalues, consequently there are infinite isosurfaces. The question that remains concerns choosing between infinite isosurfaces. Volume data is usually bounded between two scalars, such as 0 and 255, for example, which restricts the search. The user needs to choose one isovalue between these boundaries. The resulting surface isolates the area of interest. But, still, it is often the case that the scientist does not know the specific values of the volume in which he is interested. This varies depending on the purpose of the visualization.

In medicine, computer tomographies (CTs) and magnetic resonance imaging (MRI) help physicians diagnose multiple conditions in patients. These scanners extract densities in tissues and bones in a series of 2D slices. The physicians might analyze the content in each slice separately, but their composition to 3D makes a much more intuitive view of the patient. A very small abnormality in one of the slices may indicate something of uttermost importance to a possible diagnostic and further treatment. The physician is trained to see these differences, so details are very important to the visualization. By grouping these slices in 3D, the interactive navigation through multiple scalar densities shows the same abnormality with a 3D perspective. It presents the physician a clear view of the multiple layers of the volume. Figure 1.2 has an example of how this navigation works on an engine and a human body.

There is another class of techniques, known as *direct volume rendering*, which is also important and sometimes even more insightful than isosurfaces for human body investigation. It consists of casting rays through the volume from the point of view of the camera and measuring properties such as opacities or colors. However, these methods usually require fine adjustment of multiple parameters (transfer functions which adjust colors based on densities, for example). In contrast, polygonization of isosurfaces requires only a scalar (isovalue) change. Obviously, direct volume rendering can also extract isosurfaces, but

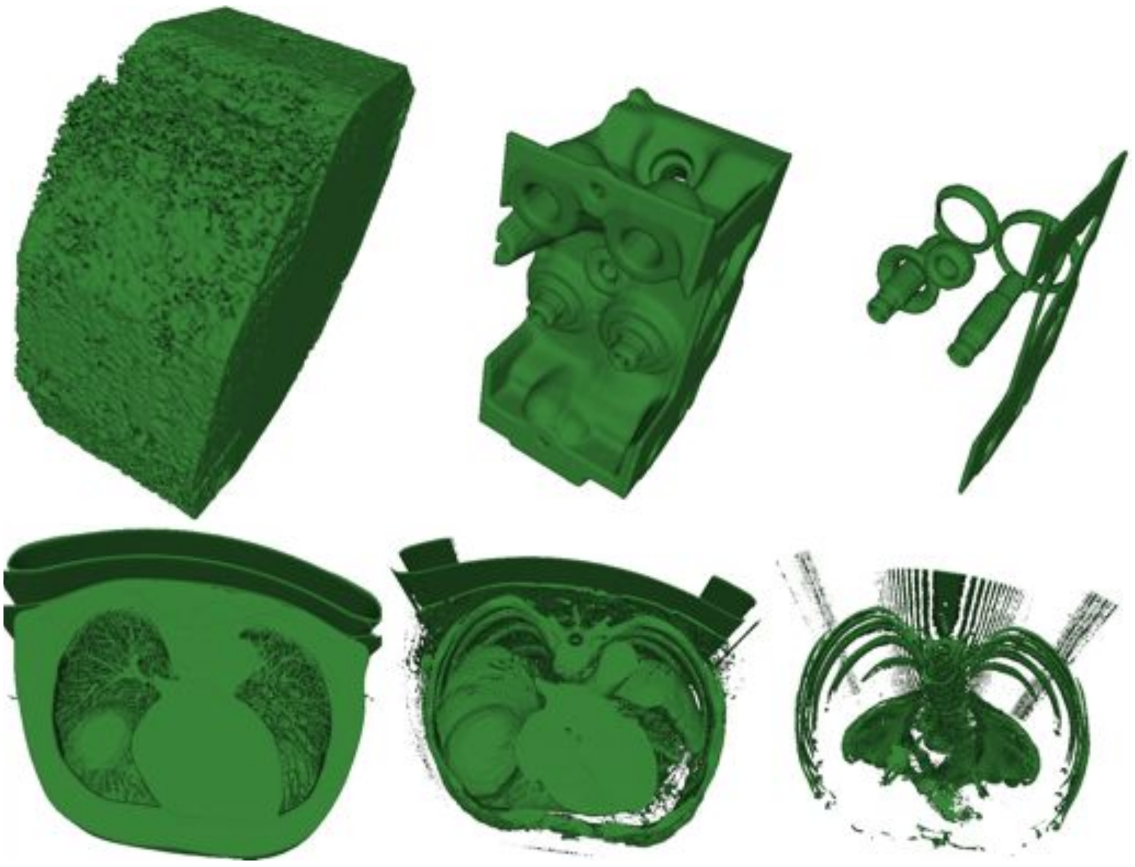


Figure 1.2: Isosurface application. Navigation through multiple isosurfaces in an engine and in a human body. The interaction is important for the visualization of the different layers in the volume data.

many applications need explicit surface information. For instance, if the extracted surface is used as input to another application, no geometric properties such as curvature, surface area and object volume can be easily computed without an explicit representation.

There are applications for isosurfaces unrelated to volume visualization. As surfaces often appear in many computer graphics fields, sometimes it is desirable to have their volumetric representation. There are many algorithms (KAUFMAN; SHIMONY, 1987; KAUFMAN, 1987; EISEMANN; DéCORET, 2006) that convert from triangle meshes into volumetric data. This process is called voxelization or 3D scan-conversion. Some operations are implemented much more easily in volumes than directly on triangle meshes.

In modeling, constructive solid geometry (CSG) operations are intuitive and fast in volume data. For example, to subtract a sphere from a set of points, the only necessary operation is to eliminate the points within the range of the sphere radius. It is common practice to voxelize the surface, perform CSG operations and afterwards bring the surface back to the triangular form with a polygonizer. The polygonizer that extracts the triangle mesh needs to be robust enough to retain details from the original surface (assuming the voxelization was robust as well).

1.2 Contributions

This thesis focuses on the use of GPUs in the polygonization of isosurfaces. Its main contribution encloses three closely related and important problems:

1. The mapping to the GPU of high-quality isosurface polygonizers.
2. The interactive polygonization of large datasets.
3. The proposal of improvements for the GPU architecture for isosurface extraction.

The only polygonization algorithms mapped to the GPU, to our knowledge, are Marching Cubes and Marching Tetrahedra (another important polygonizer) (HALL; WARREN, 1990). The featured extraction of isosurfaces, not available in either of the original versions of these algorithms, is an important contribution in this thesis. We were able to map a modified version of Dual Contouring (an approach that finds corners and sharp edges) to the GPU. Our approach to Dual Contouring approximates features in an original fashion, more appropriate for a GPU version. Details are given in section 4.2. In addition, we create a novel table-based version of Dual Contouring for improved interactivity.

Another important polygonizer implemented in the GPU in our work is Macet (DIETRICH et al., 2008). Macet improves on triangle meshes generated by Marching Cubes with one extra module than in the original algorithm. The improvement of the triangle meshes is significant and is considerably faster than in CPU. Thus, it is possible to improve triangle meshes generated by overwhelming volumes.

The interactive polygonization of isosurfaces on the GPU is usually restricted for medium-sized datasets (256^3). The extraction of isosurfaces using larger datasets, such as 512^3, is not as trivial. This type of dataset is not as large as the ones extracted by *out-of-core* techniques. *Out-of-core* algorithms offload data to disk, which is slower than the main memory and GPUs. Instead, our technique deals with datasets that fit in the main memory, but do not fit on GPU memory (usually smaller). We further evaluate the use of GPUs with a deep analysis of its bottlenecks along in the context of polygonization of isosurfaces.

Sometimes, the programmable modules have constraints that are not easily circumvented. For example, until the introduction of Directx10, no geometry could be produced inside the graphics card. In situations like this, many computer graphics researchers just try to adequate the algorithms to fit the hardware. Our work implements state-of-the-art versions on existing GPUs as well, but then we go further by using profiling tools. They are useful for the discovery of bottlenecks and proposal of improved architectures. The profiling of our techniques has detailed information about what parts of the programmable modules are used the most and what parts are not used at all.

1.3 Structure of the Thesis

This thesis is organized in seven chapters. This chapter presents context and motivation of the problem of isosurface polygonization in the GPU. It also shows our objective and contributions for the literature. Chapter two overviews common terms used in the thesis and shows important features for polygonizations. Chapter three presents the classic Marching Cubes algorithm and its variations. Chapter four presents common dual contouring algorithms and our novel approach for Dual Contouring. Chapter five describes technology and algorithms involved in GPU polygonization. Chapter six shows our results and proposed GPU improvements in the context of polygonization. Finally, Chapter seven concludes our research and discusses the future research possibilities.

2 BACKGROUND

The visualization of isosurfaces is a relatively old research topic, being almost three decades old. There are many different terms in literature that correspond to the same concept. For example, in Marching Cubes, the element that divides the grid is called voxel, cube or even cell. This chapter provides a quick reference guide (section 2.1) for the sake of clarity. Moreover, there are even worse cases in which terms are used with interchangeable meaning. Cell is used by some authors to describe the face of a cube, and not the cube itself, as it is used in this text. This further justifies a section just to give semantics to recurring terms.

This chapter also focuses on the desirable characteristics for isosurface polygonizers, such as the manifold and watertight properties. In addition, other important quality features for the polygonizations such as the approximation of the isosurface and polygon aspect ratio are included here. These are the main goals of our work and, consequently, are described in detail. Each of these properties has a different impact on applications and is later detailed.

In some applications, such as finite-element analysis, the aspect ratio of polygons is fundamental. On the other hand, if the only purpose of the application is a fast visualization, polygon quality may be traded-off for quality in the approximation. Thus, the extraction of the isosurface is completely dependent from the main purpose of the application. Implications of quality tradeoff are described below.

2.1 Glossary

In this section we present a quick reference to the technical terms used in our work. The explanations are contextualized in the field of polygonization of isosurfaces. The only purpose of this section is to clarify items used throughout the thesis, not to generalize concepts. The words are sorted[1] alphabetically and illustrated on Figure 2.1.

1. **Active edge**: Edge of a grid (regular or irregular) intersected by an isosurface.

2. **Cell**: Cube or tetrahedra (element that subdivides the structured or unstructured grid). In other literature sometimes cell refers to the face of the voxel.

3. **Cube**: Regular grid subdivision element. It is also the primitive used to divide data volumes with Marching Cubes.

4. **Cut Point**: Point where an isosurface cuts an active edge.

[1]Synonyms are not grouped. One word holds the semantics, while others just point to it.

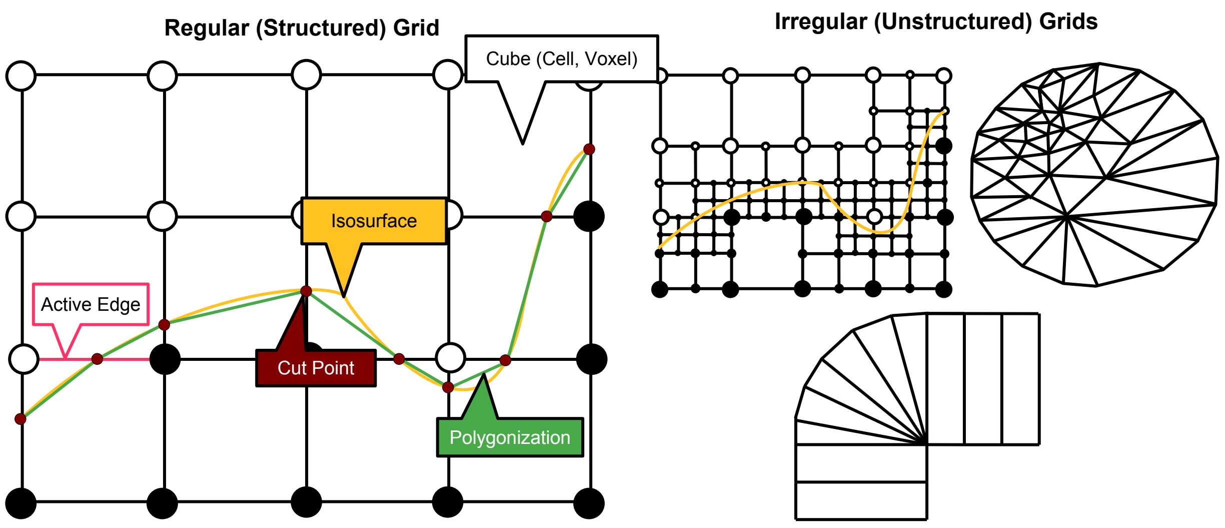


Figure 2.1: Illustration of common terms in the polygonization of isosurfaces. The figure shows a 2D case, but it can be seen as an orthogonal slice of a 3D example. Therefore, words such as cube for a square still make sense.

5. **Dataset**: Volume data.
6. **Irregular grid**: Subdivision that uses elements of irregular shapes.
7. **Isosurface**: A contour inside the data volume, where a certain property assumes a constant value, *i.e.*, $I(i) = \{x|f(x) - i = 0\}$.
8. **Isovalue**: Constant parameter that defines an isosurface within a data volume.
9. **Polygonization of isosurfaces**: Extraction of an isosurface mesh (often a triangular mesh).
10. **Regular grid**: Regular distribution of points in a discrete function.
11. **Sample**: Scalar value of the volume data.
12. **Structured grid**: Regular grid.
13. **Unstructured grid**: Irregular grid.
14. **Volume data**: Set of 3D scalar points acquired from the real world, simulations, mathematical functions or conversions from other modeling techniques. It is often referred as a discrete function.
15. **Voxel**: Cube.

2.2 Quality

Quality can be measured in different ways for the extraction of isosurfaces, with the ones most relevant to our work being presented in this section. Measurement can be done in terms of four different criteria: the polygon shape, the distance to the surface and the watertight and manifold properties. The polygon shape matters for finite element analysis and tetrahedral mesh generation, while the distance to the surface is important for almost every application, because coarse approximations might present false information. Manifold and watertight properties are important for applications such as surface simplification and collision detection, respectively.

Some of these characteristics are intrinsic of extraction algorithms, while others need extra computational modules for that purpose. These extra modules significantly impact the efficiency of the original algorithms. As one of our goals is to achieve quality and also efficiency, the aspects that impact quality are studied in detail. Thus, the following quality concepts are discussed in every polygonizer described in this thesis (sections 3 and 4).

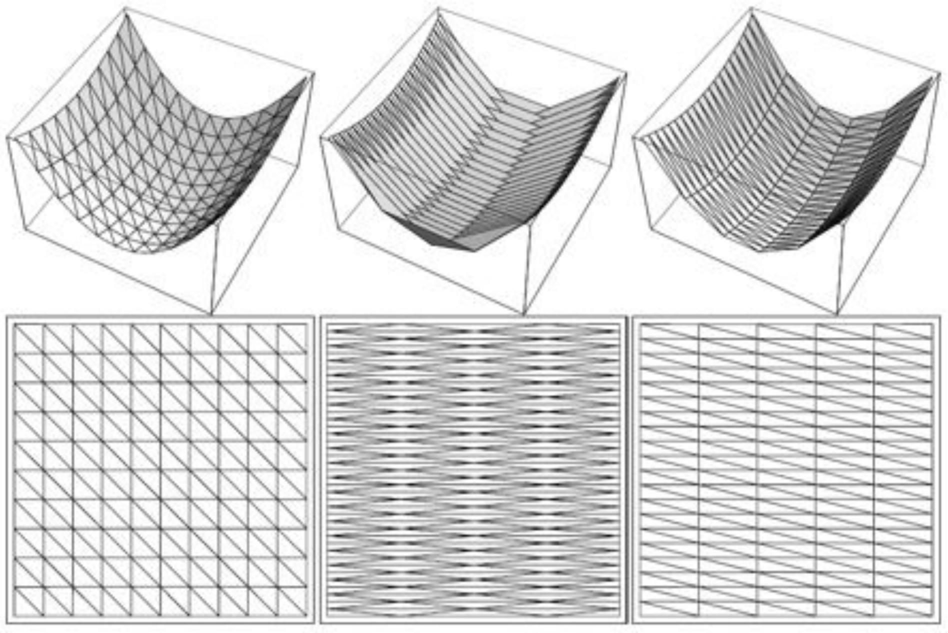


Figure 2.2: Effects of wide angles of triangles. 200 triangles with different aspect ratio were used to render a paraboloid with a piecewise linear approximation. Wider angles present worse approximation of the paraboloid. Image extracted from (SHEWCHUK, 2002).

2.2.1 Polygon Shape

There are basically two types of polygons extracted in the polygonization of isosurfaces. Triangles are commonly extracted by primal methods (section 3). Quadrilaterals (quads) are a common primitive for dual methods (section 4). Evidently, this is not a rule, since quads can be used in primal methods and triangles on dual methods. However, triangles are the simplest and most common primitive for rendering surfaces in computer graphics, thus we will analyze all polygon meshes as such.

The quality of triangle meshes is very important for our applications. Although not so relevant to isosurface visualization, it plays a crucial role in simulation procedures of finite-element analysis and tetrahedral mesh generation. The example presented in Figure 2.2 is even simpler to understand. It shows one scenario where wide angles significantly affect the approximation of a paraboloid. This type of quality refers to the triangle aspect ratio from every triangle in the surface mesh.

The simple existence of a single bad triangle in the mesh represents an ill-conditioned situation for further numerical simulation. The triangle with the worst quality often defines the quality of the entire mesh (SHEWCHUK, 2002). If this quality is important for the final application, a surface with a good average aspect ratio is not enough. The polygonization technique needs to ensure that the worst case triangle does not affect the application. If an algorithm such as Marching Cubes is used, then the triangles need to be rearranged with a post-processing algorithm, such as Laplacian Smoothing.

To compute numerical results of triangle quality (Q), the ratio between the incircle radius R_{in} and the circumcircle radius R_{out} is used. This is an often applied criterion for the purpose of measuring triangle quality.f The ratio is computed as follows: $Q = R_{in}/R_{out} * 0.5$. The scalar 0.5 (half circumcircle radius) is simply used to normalize results, scaling quality to range from 0 to 1. By using this evaluation, it is possible to compare the quality among polygonizers.

2.2.2 Distance to the Isosurface

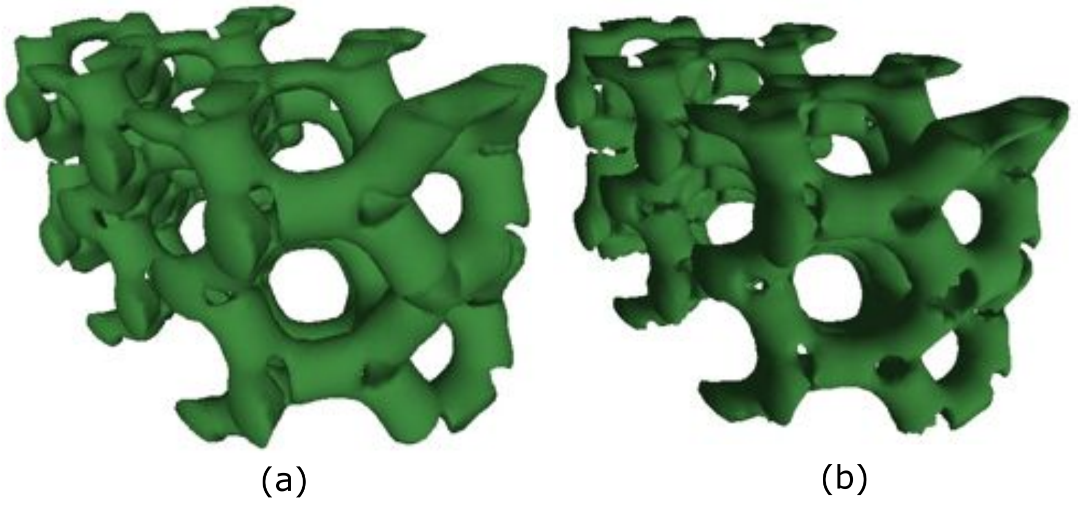


Figure 2.3: Artifacts in the silicium dataset. (a) Silicium extracted with Dual Contouring. (b) Silicium extracted with Marching Cubes. The two polygonizations represent the same isosurface, but are extracted with different polygonizers. The first is faithful to the isosurface. The second has a spiky appearance. This aliasing occurs because there are regions with sharp edges.

The polygon distance to the isosurface is an intuitive type of quality measurement. The polygonization is accurate when it is near the isosurface. Applications that would need accuracy in this respect are not difficult to imagine. If the volume data is extracted from an MRI, the physician should see a reliable approximation of the isosurface. Even when there is already some error in the data acquisition, additional error is not acceptable.

Notice that the polygon shape quality is not related with its distance to the isosurface. In many cases one quality is exchanged by the other. Bad aspect ratio polygons might represent with better precision an isosurface than good ones. It is hard to maintain good aspect ratio triangles in a complex surface. On the other hand, badly shaped triangles might just fit. For example, a long thin strip can be represented by two long triangles. Since long triangles have a very small angle, the aspect ratio is low. Only with several smaller triangles this strip can have a good aspect ratio triangle mesh. This occurs because the good aspect triangles are bounded in some angle threshold.

To compute numerical results for the distance to the isosurface is not easy. As the *real* isosurface is implicit in the volume data, its distance to the polygons is hard to measure. An application could measure this quality by ray-casting from the surface towards all directions. The closest point would indicate a relative error. This should be done from all points of the surface. However, this measurement could take too much computational time.

A common practice to measure this quality is to use Metro (CIGNONI; ROCCHINI; SCOPIGNO, 1998). Basically, the software compares the Hausdorff distance between two triangle meshes. The assumption is that one mesh represents the isosurface extreme accurately. Methods, such as Marching Cubes, might generate surfaces very close to the isosurface by using a high resolution grid. The problem is that no polygonization method really represents the isosurface with real accuracy. Samples outside the discrete function use some approximation technique, which might be a wrong assumption in the first place. Marching Cubes samples are trilinearly interpolated, which works for smooth isosurfaces.

On the other hand, if the dataset is not smooth, Marching Cubes may not be close to the isosurface. In the presence of sharp features and corners, other methods such as Extended Marching Cubes (KOBBELT et al., 2001) might better approximate the isosurface (see Figure 2.3). Thus, they might be a better choice in case Metro is used to measure the approximation. We assume that sharp features are relevant in this work, so we consider the triangle meshes from Extended Marching Cubes superior to those of MC.

2.2.3 Manifold

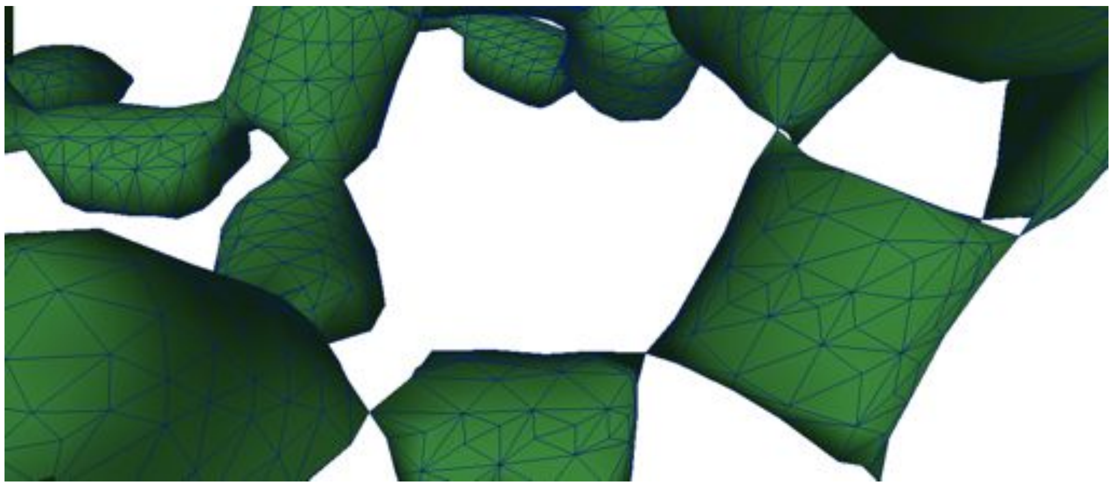


Figure 2.4: Non manifold triangular mesh. The surface has vertices that are shared by distinct parts, and locally are not topologically equivalent to a disk.

When we think about surfaces, we usually imagine the ones that are 2-manifold (or simply manifold). Surfaces composing a topological space (point-set topology) in which every point has an open neighborhood which is a disk (GUIBAS; STOLFI, 1985). Topological equivalence is less intuitive than geometrical equivalence, since it is related with the connections and not the form of the object. Topology ignores geometrical form, which means that an object might be scaled and bended and continue with the same topological connections. Thus, a manifold surface is the one in which every point of the surface can be locally flatten out into a disk-like form.

Non-manifold surfaces can be problematic. When a situation such as sharing one edge happens, surface fairing and parameterization might be compromised. Thus, the generation of manifold meshes is a desirable characteristic. There are many mathematical implications about manifold surfaces, but they go beyond the scope of this thesis.

Some polygonizers do not achieve this property directly. Dual methods are an example of that (see Figure 2.4). Sometimes disjoint surfaces remain *glued* by one point. This occurs because there is only one vertex being generated inside a voxel in which two vertices should be (SCHAEFER; JU; WARREN, 2007). In analogy to MC, this is the case of ambiguity, which is detailed in section 3.2.1.

2.2.4 Watertight

Watertight polygon meshes are surfaces without *holes*. The separation from inside to outside the surface characterizes watertight meshes. Notice that geometrical objects such as a torus have holes, but not in the polygon mesh. Figure 2.5 shows a watertight and a non-watertight triangle mesh.

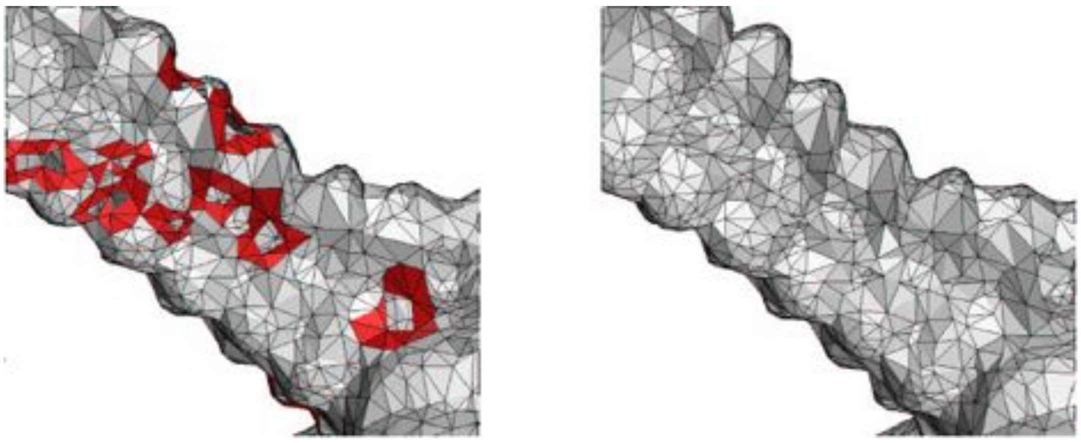


Figure 2.5: Non-watertight (left) and watertight surfaces. There are *holes* in the non-watertight surface surrounded by highlighted red triangles. Image extracted from (DEY; GOSWAMI, 2003).

Triangle meshes with incorrectness in the topology (non-watertight) are usually undesirable. There are several problems when meshes with holes are used as input. For example, when the non watertight surface is used in collision detection, an object might incorrectly trespass the missing triangles. In addition, they are not visually appealing.

Some extraction techniques in the literature need to fill holes depending of the input dataset, such as delIso (DEY; LEVINE, 2007) as reported by Raman and Wenger (RAMAN; WENGER, 2008), and even some cases of Marching Cubes (WILHELMS; GELDER, 1990) (if the look-up table is incorrectly configured). They lack the production of a closed triangle mesh in particular situations. Marching Cubes might produce inconsistencies in the ambiguity case, which is explained in Section 3.2.1.

3 PRIMAL METHODS

Isosurface polygonization is a well-studied problem, but still a focus of interest nowadays. Since the pioneering work of Udupa (HERMAN; UDUPA, 1982), several (and significantly different) approaches for extracting polygon meshes from implicit surfaces were proposed. These approaches can be categorized into three broad classes: (1) methods based on domain subdivision, (2) pseudo-physical methods and (3) surface tracking methods. Our work is focused on methods based on domain subdivision, since its underlying approach usually leads to high-performance polygonizers.

Such methods follow the *divide-and-conquer* paradigm, that subdivides the domain of the function f ($f : R^n \rightarrow R$) into a structured (or unstructured) grid. The grid is composed by a set of cells, which are processed independently. The isosurface inside each cell is approximated by a set of polygons, in a way that, when combined for all cells, forms a watertight and manifold mesh. Thus, all of the algorithms presented in this section have both properties.

We present three contouring techniques in this chapter. All of them are important for the understanding of our thesis goals. Cuberille is historical and helps the understanding of divide-and-conquer algorithms. Marching cubes is the standard and used for comparison with novel techniques. Finally, marching tetrahedra is as important as MC in the context of GPU polygonization of isosurfaces.

3.1 Origins

At the beginning of the eighties, cuberille was a popular choice in isosurface extraction. Computer graphics was severely constrained by memory and computer processing power. The algorithm is extremely simple, but robust and might be used for fast isosurface visualization.

The subdivision elements in the cuberille technique are voxels (cubes). Consequently, this technique works for polygonization in uniform grids. The algorithm for the polygonization is very simple. Each cube is sampled at its center. If the sampled value is inside the isosurface (lower than the isovalue), then that cube is opaque. Otherwise, if the sampled value is outside the isosurface, that cube is transparent. The result is a set of cubes that approximate the isosurface. However, note that this forms a solid object, and we are interested in a surface. Instead of just setting the cube as opaque or transparent, we just render the faces that are shared by cubes inside and outside the isosurface, as shown in Figure 3.1.

The algorithm has the advantage of being extremely fast, as a consequence of its simplicity. It also produces high-quality polygon shapes (the mesh may be formed by quads or even triangles). However, the main problem is obvious: the surface is coarsely

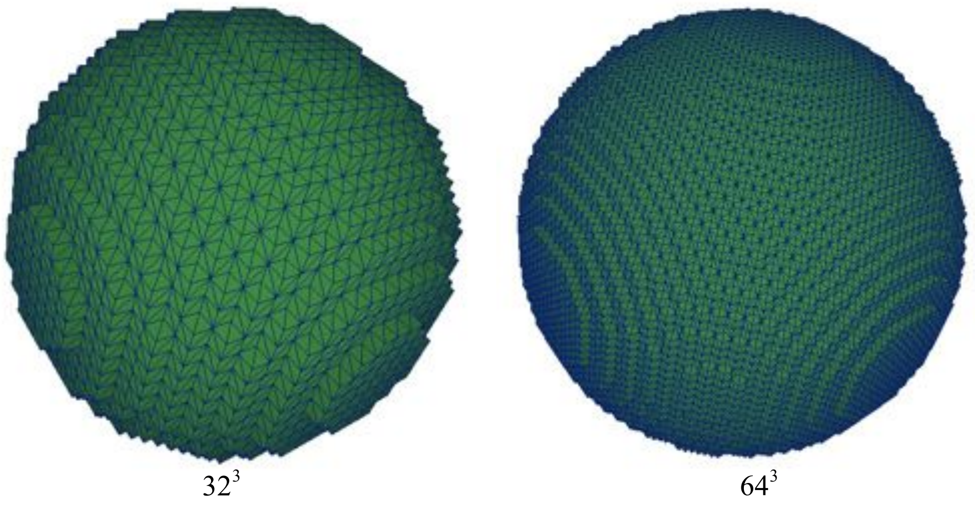


Figure 3.1: Sphere extracted with Cuberille at two different resolutions. Historical approach to polygonize isosurfaces. Refinement results in better approximation, but doubles the storage requirements for each dimension.

approximated. The aliasing in the surface appearance occurs because there is no fine adjustment of the polygon vertices. This continues visible even at high resolutions, which increases the storage costs.

There were proposed solutions for the aliasing, such as using an adequate shading technique (CHEN et al., 1985). With the inclusion of smooth shading, the resulting images are more interesting. The polygon normals are bended following the normal direction of the isosurface. This solution is good for fast and reasonably faithful visualizations. However, the surface is still coarsely approximating the real isosurface.

The following algorithm, Marching Cubes (MC), is dual to cuberille. Each face of cuberille is a point in Marching Cubes. But why do we call this chapter Primal, and not Dual Methods? Because MC is usually referred to as primal, while other methods similar to cuberille are called dual contouring algorithms (details about dual contouring algorithms are presented in chapter 4).

3.2 Marching Cubes

Marching Cubes is a fundamental algorithm in the polygonization of isosurfaces. The algorithm is widely accepted by the visualization community, for its simplicity and robustness. It is also referenced many times in other contexts, such as in modeling surface reconstruction, as previously mentioned (section 1.2).

The algorithm divides the volume into a grid of regular cells (Figure 3.2a), such as in the Cuberille. It operates on two fundamental steps: detection of *active cells* (cells crossed by the isosurface) and generation of triangles inside each active cell. This separation is an important characteristic for a pipeline implementation (section 5.1).

At each vertex cell, the scalar value associated with the location of this vertex is evaluated from the data volume, compared against an input isovalue, and classified as either inside or outside the isosurface. Every edge of this grid that has its two endpoints with opposite classifications is assumed to cross the isosurface (Figure 3.2b shows red vertices over the active edges).

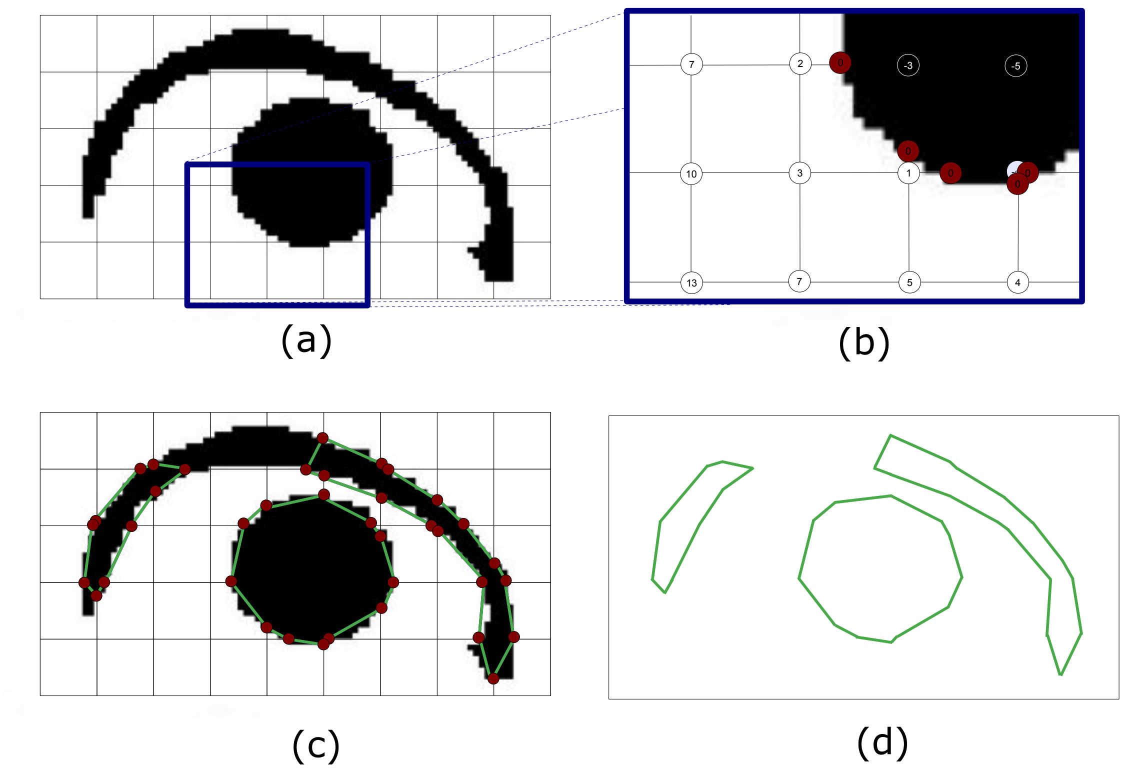


Figure 3.2: Marching Cubes 2D algorithm. (a) The isosurface here is represented as the computer graphics group logo. The grid is a uniform voxelization of this logo in very low resolution. (b) The red points are linear interpolations along the active edges, where the isosurface cuts. (c) The green edges show the resulting polygonization connecting the red points. (d) The resulting low-resolution approximation of the computer graphics group logo.

The limited combination of output classifications for the eight vertices of a given cell (a finite set of $2^8 = 256$) is encoded in a table that defines the number of triangles and their topology. This configuration of the cell is used as an index to access directly the triangulation case in a table-driven fashion. The 3D classification of topologically distinct cases is shown in Figure 3.3. Crossings can be found in several ways: assuming that the point is exactly in the middle of the edge; doing a linear interpolation along the edge; doing a bisection algorithm in the active edge; etc. The original MC proposes the use of linear interpolation, which is a fine approximation and not as costly as using bisection (useful if the underlying isosurface has a coarse granularity).

The first step of MC is related to the algorithmic efficiency. The complexity of the *marching* in the cubes is $(O(n^3))$. As the number of empty cells is usually very large, there are many works which attempt to improve this first step (SHEN et al., 1996; BAJAJ; PASCUCCI; SCHIKORE, 1996; CIGNONI et al., 1997). There is also the alternative to operate in a view-dependent manner (GREGORSKI et al., 2002). The view-dependent approach marches through the cubes with respect to the camera, avoiding back-faced triangles. However, we are interested in the extraction of the whole surface and not just its fast visualization.

The second step is related to the quality of the final mesh. The shape of the triangles is dependent on those pre-defined cases. There are different possibilities in the triangulation

(DIETRICH et al., 2008), which impact the quality of the triangle mesh. The quality of the approximation is defined in this step, since here the isosurface intersections are computed. MC surfaces are accurate, in the case of smooth isosurfaces.

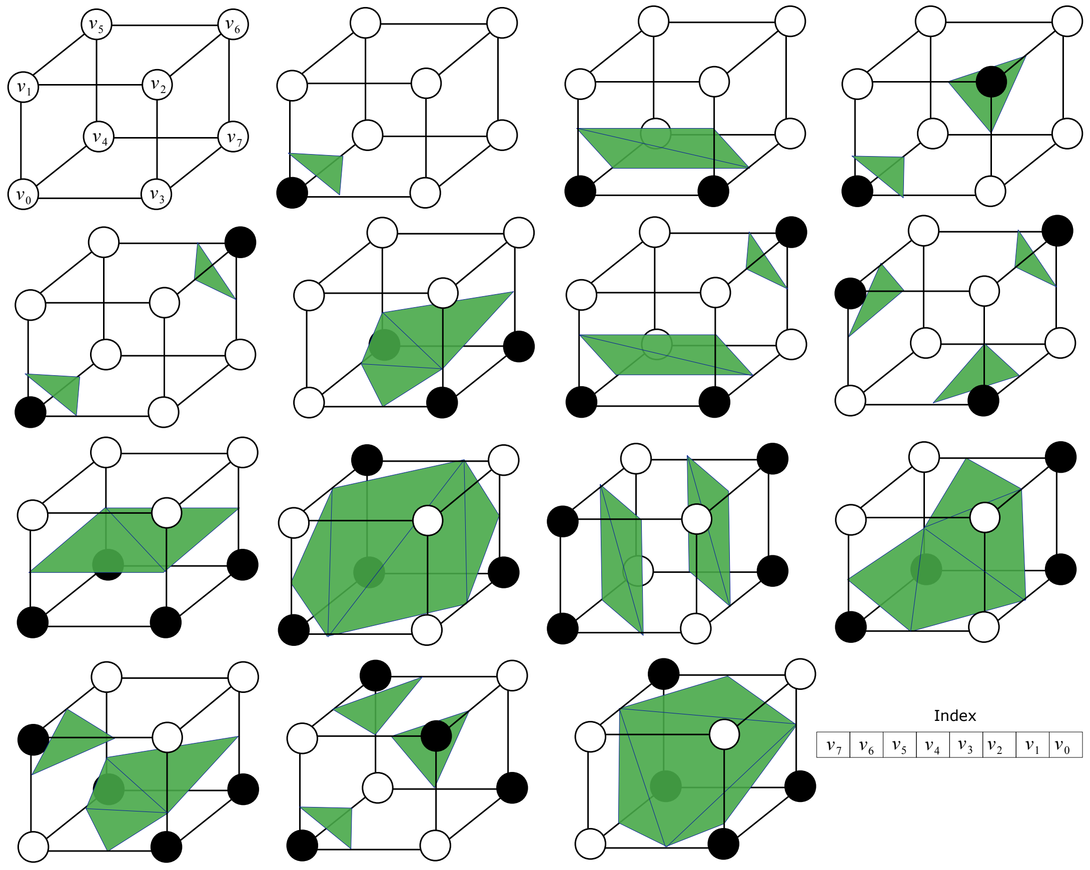


Figure 3.3: Original Marching Cubes output classification cases with topologically distinct triangles. Black dots are samples from inside the isosurface. White dots are outside. The configuration case can be computed as an eight bit index $v_7v_6v_5v_4v_3v_2v_1v_0$ and used as a key for the identification of the topology of the triangles.

3.2.1 Shortcomings

In the previous section, we enumerated the strengths of Marching Cubes. This section overviews its main shortcomings and shows common solutions that can be found in the literature. Since our work focus on quality, it is important to show where MC fails. On the other hand, this section also shows that the problems are only relevant for particular applications.

One of the first and most studied problems is the ambiguous face (Figure 3.4(a) in the topology table). This might produce topological inconsistencies in the production of the triangle mesh (WILHELMS; GELDER, 1990). If the table specification is not defined with caution, two ambiguous faces might end up with opposite table configurations. Consequently, "holes" could appear and the triangle mesh has lost its watertight property, leading to several problems, as discussed in section 2.2.4. Besides, even if the table is consistently specified, the polygonization may present an incorrect isosurface (downside for the quality in approximation). The ambiguous face can separate or connect two different parts of the isosurface, depending on the specified topology.

One approach to solve this is to extend the MC table to include these two cases. After an ambiguous face case is identified, the algorithm needs to decide which one is the correct case. Nielson and Hamman (NIELSON; HAMANN, 1991) did this by bilinearly interpolating over the ambiguous face (Figure3.4(b)). The intersection of the asymptotes decides which is the correct topology case. Another useful and even simpler approach is estimating the center value of the ambiguous face (WYVILL; MCPHEETERS; WYVILL, 1986) (Figure3.4(c)). The sample is calculated as the average of the vertex values of the face.

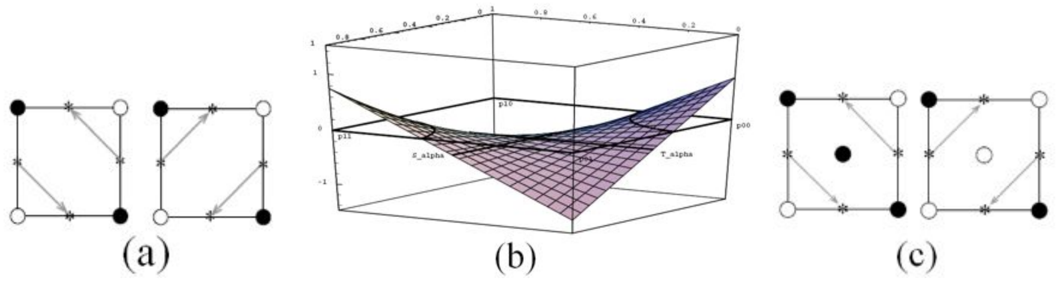


Figure 3.4: Ambiguity problem. (a) The ambiguous face and two proposed solutions: (b) The asymptotic decider. Bilinear interpolation over the face. (c) Estimate a point value in the middle of the face. Image extracted from (WUENSCHE, 1997).

Another problem is related to finding a given grid resolution that allows the isosurface to be correctly approximated by a piecewise linear approximation. Solutions often consider adding resolution only in areas that require so, therefore leading to an adaptive subdivision. Usually, a subdivision criterion tries to minimize the distance between the linear approximation and the isosurface. It is not possible to have different neighboring cube sizes in the original MC, as can be seen on Figure 3.5.

Examples of algorithms that address this issue are Dual Contouring (section 4.2) and the method proposed by Bloomenthal (BLOOMENTHAL, 1988). The first is intrinsically adaptive for its dual nature, but it requires that the initial grid is maximally collapsed. This means that the adaptive subdivision simplifies the surface, but does not refine it. The latter uses octree subdivision to track the surface, but it is rather complex to implement.

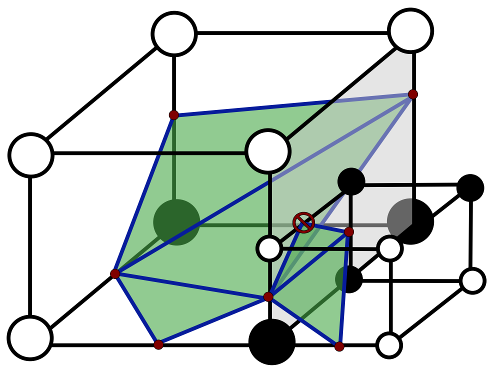


Figure 3.5: Adaptive subdivision problem with different sized voxels. The figure shows an issue in the use of the original MC with an octree. The polygonization may not be watertight.

Another problem is the aspect ratio of the triangles generated by MC, which is one of the focuses of this work. There are many works that center the attention in the quality and correctness of the triangles (LEWINER et al., 2003; LOPES; BRODLIE, 2003; NIELSON, 2003). An interesting example of the latter is Macet (DIETRICH et al., 2009, 2008), an algorithm based on the notion of edge transformations. By allowing the position of MC edges to be placed in more convenient locations, their proposal achieves much better triangle quality and requires minor modifications in the original MC code.

A problem we find even more interesting is the featured extraction of isosurfaces. As MC is meant for smooth isosurfaces, corners are often smooth. Many datasets have normal information (Hermite data). These normals open the possibility of finding features, such as sharp edges and corners. Kobbelt (KOBBELT et al., 2001) extended MC into a featured polygonizer. The method explicitly tests for sharp features, and if it is the case, uses a triangle fan to render the edge or corner. If there is no sharp edge, the table of MC is used. The central vertex from the fan is located at the minimizer of the quadratic error function (solved by Singular Value Decomposition):

$$E[x] = \sum_i (n_i \cdot (x - p_i))^2 \tag{3.1}$$

The final problem we want to raise is the computational cost. If one method does not have the previously mentioned problems, it will probably take much more time to extract the surface than the original Marching Cubes. This is an important reason why to implement divide-and-conquer polygonizers on parallel hardware, such as the GPU, and decrease its complexity.

3.2.2 Edge Transformations (Macet)

Macet (DIETRICH et al., 2008) (Marching Cubes using Edge transformations) is a technique that improves the triangle quality generated by Marching Cubes (optionally Marching Tetrahedra). Basically, it perturbs the position of the active edges in a way that avoids degenerate polygons. The algorithm is based on the previous knowledge of situations in which bad triangles happen, such as near corners (Figure 3.7).

The technique uses an extra computational module for MC. The module is situated after the discovery of active edges and before linear interpolation. It moves the position of the active edges following one of two approaches (see Figure 3.6). Both approaches improve the triangle quality, but one path may be better than the other at particular situations. Thus, they are combined to form the final mesh. The two approaches are described as follows:

- **Gradient**: The edge points ($\overline{V_i V_j}$) move along the gradient

$$\nabla f(x, y, z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right) \tag{3.2}$$

  In which the partial derivatives of f are estimated by central differences along the axis (the size of the lattice L is used):

$$\begin{aligned} g_x &= f(x + L_x, y, z) - f(x - L_x, y, z), \\ g_y &= f(x, y + L_y, z) - f(x, y - L_y, z), \\ g_z &= f(x, y, z + L_z) - f(x, y, z - L_z), \end{aligned}$$

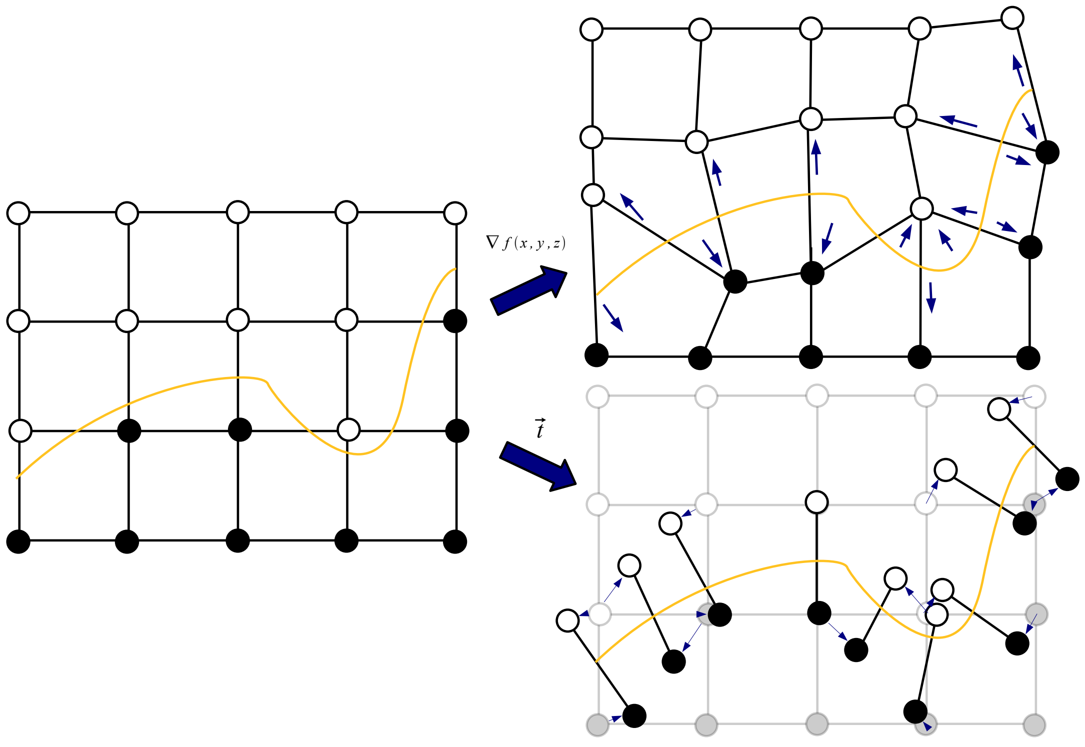


Figure 3.6: MACET 2D gradient and tangent edge transform examples. The gradient outside the isosurface is used to move the grid (upper-right). The tangent transform moves the edges along tangent trails disconnecting the grid, but retaining mesh topology (lower-right).

$$\nabla f(x,y,z) \approx \|(g_x, g_y, g_z)\| \tag{3.3}$$

This gradient estimation works for every point of the scalar field. It is possible now to move both edge points along the gradient path.

$$\begin{aligned} V_i &= V_i + \alpha * \nabla f(V_i) && (3.4)\\ V_j &= V_j + \alpha * \nabla f(V_j) && (3.5) \end{aligned}$$

There is one constraint not included in the above formulae. In order to maintain differentiable meshes, it is necessary to test whether one edge point crossed the isosurface. If this is the case, the step is discarded and the gradient moving stops. Furthermore, α is used to control the size of the step.

- **Tangent**: The edge points ($\overline{V_iV_j}$) move along a tangent trail in order to leave the edge as close as possible to becoming perpendicular to the isosurface. In the case the edge is active, the points are moved using the local coordinates ($\vec{b}$ stands for binormal):

$$\begin{aligned} \vec{b_i} &= \nabla f(V_i) \times (V_j - V_i), \\ V_i &= V_i + \alpha * (\vec{b_i} \times \nabla f(V_i)), && (3.6)\\ \vec{b_j} &= \nabla f(V_j) \times (V_i - V_j), \\ V_j &= V_j + \alpha * (\vec{b_j} \times \nabla f(V_j)) && (3.7) \end{aligned}$$

The algorithm has the same constraint as the gradient transform. The edge is not supposed to cross the isosurface. The tangent trail also uses α to control the size of the step. Equation (3.3) is used to estimate normals.

After extracting two meshes using both approaches, the combination of both is done. A quality criteria evaluation is established, such as the one previously presented on section 2.2.1. Macet uses a radii ratio average (PÉBAY; BAKER, 2003) of the triangles that share each vertex. The vertices with the best average are chosen between the tangent and gradient transforms. This is done in multiple passes, since the evaluation changes after vertices of these different meshes are chosen. This iteration stops when the best configuration of vertices is reached. A united version of Macet was described recently that does not require this iterative process (DIETRICH et al., 2009).

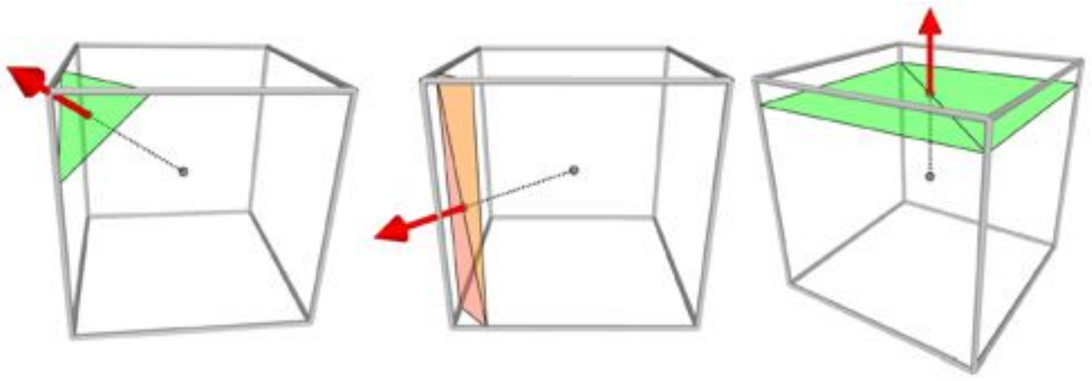


Figure 3.7: Isosurface trespassing voxels on different positions and associated triangle quality. Image extracted from (DIETRICH et al., 2008).

3.3 Marching Tetrahedra

Marching Tetrahedra (HALL; WARREN, 1990) is a common alternative to Marching Cubes. As the element of the grid subdivision is a tetrahedra, it works for isosurface extraction in unstructured grids. Its main purpose is the polygonization in such scenarios, working for physics simulations and adaptive subdivisions. In our experiments we use regular grids, so Marching Tetrahedra is applied as such. The basic difference is that the structured grid is divided into tetrahedra.

Each voxel can be separated into five or six tetrahedra (see Figure 3.8). Commonly, the best choice for the subdivision is six, because the polygonization gets smoother. Dividing the cube into five tetrahedra may lead to spiky surfaces (WUENSCHE, 1997). In many cases, this means that the polygonization is far from the isosurface.

The algorithm has the same construction of MC, so our description is brief. The first step is the detection of active tetrahedra. Then each different tetrahedra configuration ($2^4 = 16$) of active edges is consulted in a table-driven fashion for further triangulation. Even though the algorithm is practically the same as MC, the tetrahedral subdivision has important implications.

There are some advantages of MT over MC. First, MT has no face ambiguity. The tetrahedra face is composed of three vertices, forming triangles. There is only one way to triangulate the different tetrahedral configurations. In addition, the topology table is

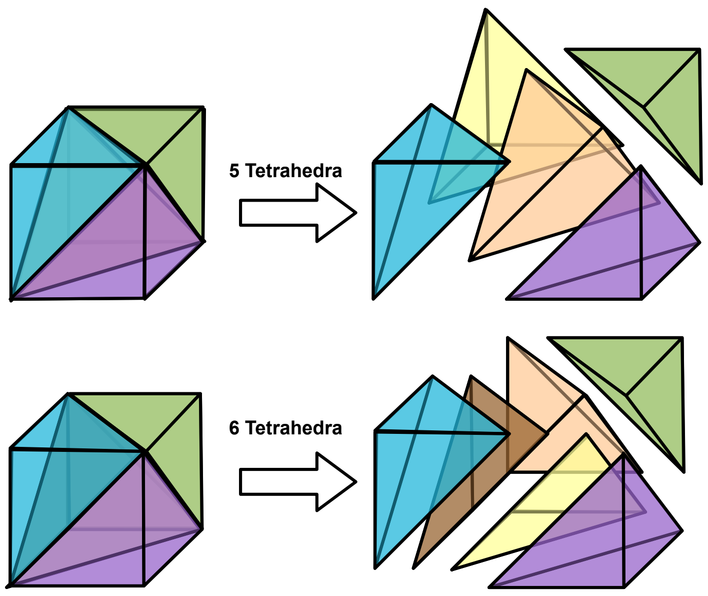


Figure 3.8: Cube subdivision. Cubic cells might be divided into five or six tetrahedra.

smaller than the one in MC, having only $2^4 = 16$ cases. Furthermore, MT can be implemented with different grid resolutions. This allows the isosurface to be correctly approximated by a piecewise linear approximation. However, the implementation is relatively complex.

The main disadvantages of MT are both quality and the number of triangles it generates. MT generates about 100 to 150% more triangles (NING; BLOOMENTHAL, 1993) than MC. This is very problematic for the storage of triangle meshes. These triangle meshes are even worse shaped than the ones in MC. In addition, the tetrahedral subdivision of regular grids is undesirable for the isosurface approximation (SNOEYINK, 2006).

Both triangle quality and number of triangles issues are improved by vertex clustering (post processing). The Regularized Marching Tetrahedra (TREECE; PRAGER; GEE, 1999) technique uses this strategy. The differences of the original MT to RMT can be seen on Figure 3.9. Of course, this implies in higher computational costs, which compromise the isosurface extraction speed.

3.4 Span Space

As previously mentioned, the most expensive step in the polygonization of isosurfaces is the detection of active edges. In MC, every cube from the uniform grid has eight vertices, each sampled once (it could be simplified to 4 vertices, in a more elaborate neighbor-dependant implementation). This results in the dataset being sampled 8 times per voxel. For instance, if the dataset has 512 samples per axis, the dataset is sampled 8 x $(512 - 1)^3 = 1\ 067\ 462\ 648$ times. As the isosurfaces are sparse in the volume, the best way to accelerate any polygonizer is to avoid cells that are not intersected by the isosurface.

Improving active cell detection is commonly done in pre-processing stages, which organize the access to the domain of function f according to a predefined isovalue. The Span Space structure is a spatial hashing with cells organized in a 2D map, based on the minimum and maximum values of f at cell vertices. The map is constructed in such way

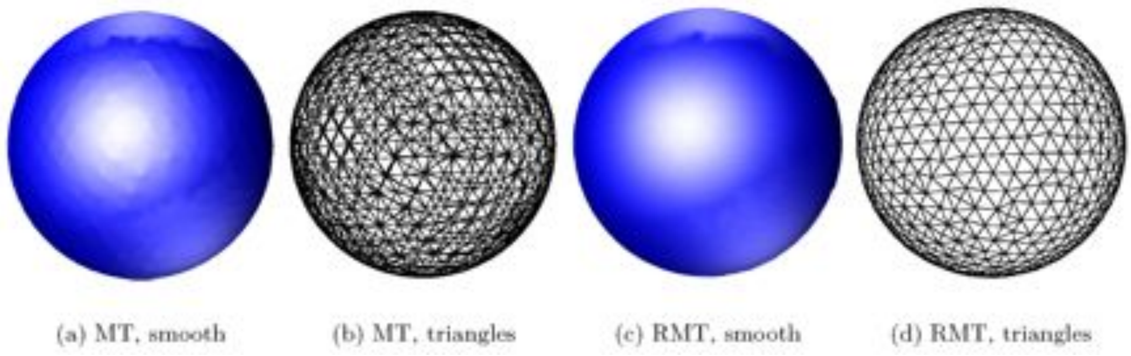


Figure 3.9: Regularized Marching Tetrahedra comparison with Marching Tetrahedra. Gouraud Shading (GOURAUD, 1998) (per vertex illumination) used to show interpolation problems. (a) Shaded MT. (b) MT with wireframe rendering addition. (a) Shaded RMT. (d) RMT with wireframe rendering addition. Image extracted from (TREECE; PRAGER; GEE, 1999).

that active cells corresponding to any isovalue are constrained to a subset of the map (see Figure 3.10), which are easily determined from the isovalue used.

Two common data structures to handle the data and provide the queries are KD-trees (LIVNAT; SHEN; JOHNSON, 1996) and interval trees (CIGNONI et al., 1997). The basic difference is that KD-trees exchange efficiency for memory. Interval trees might achieve optimal results. The worst case complexity for the interval tree search is $\theta(k + \log n)$, where k is the number of queried elements and n is the total amount of elements. On the other hand, KD-trees have this complexity of $\theta(\sqrt{n} + k))$. In addition, there is a Span Space version that is parallel efficient. This technique, called ISSUE (SHEN et al., 1996), uses a uniform grid to subdivide the space.

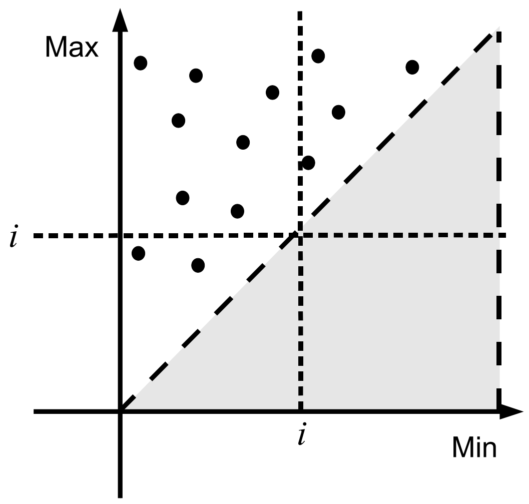


Figure 3.10: Span space. The mapped position of the voxels (black dots of the figure) in maximum and minimum space simplifies the search of the voxels with active edges into a geometrical query. The isovalue i is used for the composition of the subset of this map.

Classifying the voxels by groups of maximum and minimum is a very effective way to avoid uncut voxels. If the isosurface is sparse, this is a very elegant and efficient approach.

However, all of the approaches have a significant preprocessing time ($\theta(n \log n)$) and have large memory ($\theta(n)$) requirements. This sometimes is a harsh constraint for on-the-fly volume extractions and massive volumes, respectively.

3.5 Summary

This chapter revisited two common algorithms in the literature that polygonize isosurfaces in primal space. We started with Cuberille, which is a historical dual contouring algorithm and important for the understanding of Marching Cubes. Then we detailed Marching Cubes, an algorithm which is the basis for comparison for any isosurface extraction technique. MC was exhaustingly explored through its strengths and shortcomings. We also presented a technique for unstructured grids, Marching Tetrahedra. MT is important for later discussions of GPU based implementations. Finally, we discussed the Span Space approach for the algorithmic improvement of efficiency in every polygonizer of isosurfaces.

4 DUAL METHODS

Dual contouring algorithms place vertices inside cells instead of on grid edges. Cut points in the active edges become surfaces in these dual versions. In the case of uniform grids, the surface is constrained by the four voxels that are adjacent to this edge. If the surface is a full octree, the polygons are quads. If there are different size voxels, they are triangles.

This type of approach has some advantages, like improving the shape of the polygons and better approximating the surface. Both advantages happen because we have more freedom where to place points. Instead of interpolating through one edge, as in primal methods, the vertex has the entire voxel space to be positioned into.

We present two contouring techniques in dual space that generate watertight surfaces. Surface Nets is important for the understanding of the reasons why one should apply a dual contouring algorithm. Dual Contouring goes steps ahead, looking for even more accurate surfaces, by polygonizing featured isosurfaces. However, both original algorithms have the problem of not generating manifold meshes in the case of an ambiguous cell, which leads to the problems described in section 2.2.3. We assume input volumes sufficiently refined, so that there are no ambiguous cases.

4.1 Surface Nets

Surface Nets (SN) (GIBSON, 1998) appeared as a higher-quality alternative to Marching Cubes. The main purpose of the technique is achieving a better approximation of isosurfaces. It considers the existence of thin details in the isosurface, while many methods assume that the function is smooth.

The algorithm consists on creating *nets* around the active edges. These *nets* are polygons that converge to the isosurface with the help of pseudo-physical smoothing algorithms. The approximation occurs by reducing the energy on the connections (vertices) of the nets according to their neighbors. They move iteratively towards the position equidistant between its adjacent neighbors. The surface does not shrink into a point by using one constraint. The connections of the *net* are kept inside the cell.

The liberty to place connections is responsible for a polygonization with less aliasing. With an appropriate grid resolution, the polygonizations of SN retain details, especially important in MRIs and CTS. However, the approximation does not rely on any fine approximation of the isosurface. The vertices are kept because of the uniform grid positioning, without the underlying isosurface information.

4.2 Dual Contouring

Dual Contouring (DC) is a method that refines the accuracy of Surface Nets and works for adaptive grid resolutions. It is a hybrid between Extended Marching Cubes (KOBBELT et al., 2001) and Surface Nets. The first has the characteristic of finding sharp edges based on the point normals of the active edges. The latter has a dual-space nature. This combination leverages the good characteristics from both methods.

The algorithm consists on finding the active edge and creating a quad with its four points in the center of the four voxels sharing the edge, as in Surface Nets. After that, it positions the vertex in the isosurface feature using information of position of the cut points (as MC does) and also their normals. Similarly to the Extended Marching Cubes (EMC) feature sampling.

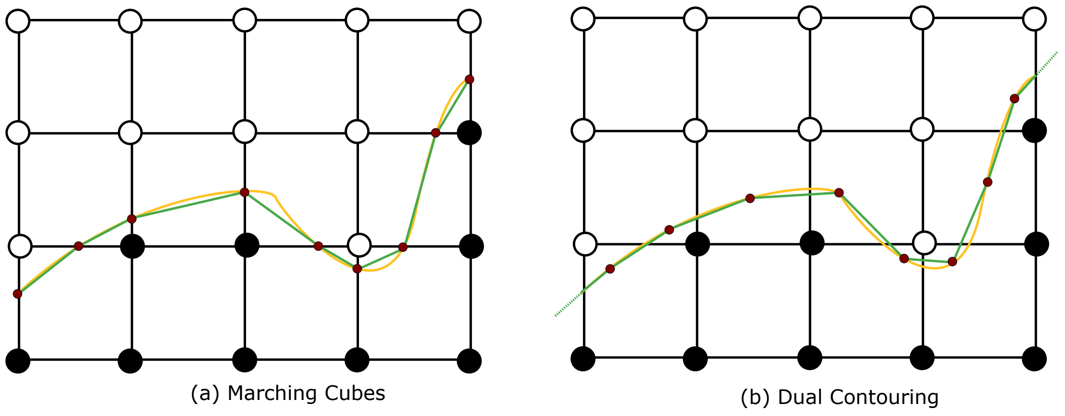


Figure 4.1: An isosurface being polygonized in primal and dual contouring. The vertices are positioned in the edges in (a) and in the QEFs in (b).

Vertex positioning is done by solving the Quadratic Error Function described in equation 3.1. QR decomposition is used in Dual Contouring instead of Singular Value decomposition (EMC approach). Ju *et al* showed that SVD needs double precision to minimize the QEF in acceptable error bounds. This impacted significantly the memory requirements for their application (doubled it). Therefore, they traded efficiency of SVD for smaller memory requirement of QR decomposition.

DC can adaptively simplify the generated meshes. It first extracts the most refined version in an octree, and creates interior nodes by adding QEFs from the leaves. The interior nodes whose QEFs have not surpassed a given error tolerance are collapsed into leaves.

The natural adaptive subdivision of this approach is an advantage of DC over most polygonizers. Since most isosurface extractors have a large memory demand, DC is an interesting alternative. In addition, there is implicit feature identification. Both characteristics are probably the most important features of DC. However, its implementation is not as trivial as MC. Both SVD and QRD often require complex computations for the vertex positioning of polygons, therefore we created a novel iterative approach that is explained in the next section.

4.2.1 Particle-based feature approximation

This section introduces our algorithm for the vertex positioning in Dual Contouring. It is an alternative to the ones presented by Ju *et al* (JU et al., 2002) and Kobbelt *et al*

(KOBBELT et al., 2001). We did not use decomposition approaches because they have expensive computations for the GPU, such as matrix diagonalization. As one of our goals is the interaction through the isosurfaces, we created a novel approach. It is based on the intuition of the description given by Kobbelt, which states why MC does not find sharp features. The minimizer of the QEF is replaced by our particle-based method, which, iteratively, moves the quad points (referred and treated hereafter as particles) towards the isosurface.

The particles start at a mass point $\bar{C}$ of the cell, calculated from the arithmetic mean of the active edge intersection points (P_i). This process reduces the number of iterations of the particle and is a good hint of the isosurface location(SCHAEFER; WARREN, 2002).

$$\bar{C} = \sum_{i=1}^{n} \frac{P_i}{n} \tag{4.1}$$

The next step is to find the force $\vec{F}$ that starts moving the particle from $\bar{C}$ towards the isosurface (Figure 4.3). Since the data used is Hermite, the normals of cut points are necessary. We use the gradient of the scalar field to approximate the normals ($\vec{n_i} = \nabla f(P_i)$).

The force $\vec{F}$ is generated by a trilinear interpolation of the forces $\vec{F_0}$ to $\vec{F_7}$ located at the grid points $\vec{V_0}$ to $\vec{V_7}$ driven by the centroid $\bar{C}$. These forces are calculated as the sum of the vector distances to the planes defined by all pairs $\vec{n_i}$ and P_i (intersection points from the voxel):

$$\vec{F_k} = \sum_{i=1}^{n} (\vec{n_i}, (P_i \cdot \vec{n_i})) \cdot S * \vec{n_i} \tag{4.2}$$

in which $S \equiv (\widehat{L} \cdot x = -p)$ represents a plane in Hessian normal form, the grid lattices L of the active edge used as normal, $w = 1$ and the point p at origin of the coordinate system. When the eight forces of the voxel have been calculated, we get the force $\vec{F}$ by trilinear interpolation:

$$\begin{aligned}
\vec{F_{l_1}} &= \left(1 - \left(\frac{\bar{C}_x}{L_x}\right)\right) * \vec{F_0} + \left(\frac{\bar{C}_x}{L_x}\right) * \vec{F_3}, \\
\vec{F_{l_2}} &= \left(1 - \left(\frac{\bar{C}_x}{L_x}\right)\right) * \vec{F_4} + \left(\frac{\bar{C}_x}{L_x}\right) * \vec{F_7}, \\
\vec{F_{l_3}} &= \left(1 - \left(\frac{\bar{C}_x}{L_x}\right)\right) * \vec{F_1} + \left(\frac{\bar{C}_x}{L_x}\right) * \vec{F_2}, \\
\vec{F_{l_4}} &= \left(1 - \left(\frac{\bar{C}_x}{L_x}\right)\right) * \vec{F_5} + \left(\frac{\bar{C}_x}{L_x}\right) * \vec{F_6}, \\
\vec{F_{b_1}} &= \left(1 - \left(\frac{\bar{C}_y}{L_y}\right)\right) * \vec{F_{l_1}} + \left(\frac{\bar{C}_y}{L_y}\right) * \vec{F_{l_2}}, \\
\vec{F_{b_2}} &= \left(1 - \left(\frac{\bar{C}_y}{L_y}\right)\right) * \vec{F_{l_3}} + \left(\frac{\bar{C}_y}{L_y}\right) * \vec{F_{l_4}}, \\
\vec{F} &= \left(1 - \left(\frac{\bar{C}_z}{L_z}\right)\right) * \vec{F_{b_1}} + \left(\frac{\bar{C}_y}{L_z}\right) * \vec{F_{b_2}}
\end{aligned} \tag{4.3}$$

Particles move along the force $\vec{F}$ with Euler integration. Best results were obtained using 5% of $\vec{F}$ in all our volumes. Fewer terracing artifacts and a better polygonization are

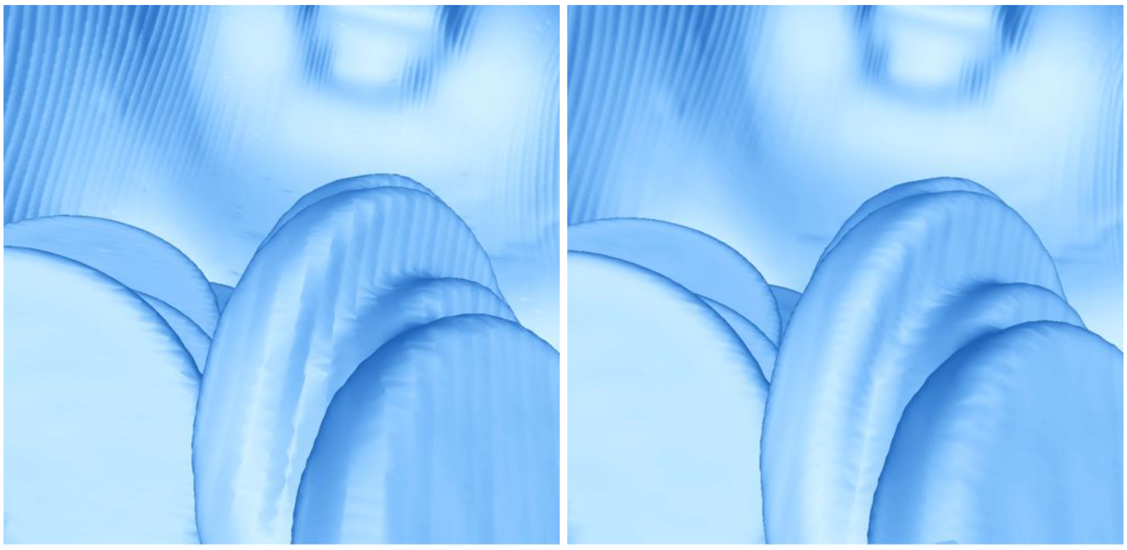


Figure 4.2: Dual Contouring fewer terracing artifacts. Marching Cubes (a) and higher quality Dual Contouring (b) using Pig dataset. Fewer terracing artifacts and a better approximation of the polygonization are presented.

presented in Figure 4.2.This new position ($\bar{C}' = \bar{C} + (c * \vec{F})$) is used in the second iteration as input to the trilinear interpolation (recalculate $\vec{F}$). This procedure is repeated for a fixed number of steps or until the particle converges (threshold driven) to the isosurface. Our method does not allow that the particle leaves the voxel by clamping its position, in a similar fashion of SN.

This iteration is responsible for a good approximation of the isosurface, as well as finding sharp features. The shortcoming from our approach is no longer having the mathematical guarantee that the method finds the feature. But the real feature is hardly correct using SV or QR decompositions, because the feature might not be sharp or even the gradient samples (normals) might be noisy.

4.2.2 Table-Driven Geometry Specification

The second modification we propose for the efficiency improvement of the original DC is a table-driven geometry specification. The table presents a description of the different configuration cases for each cell of the full octree (all nodes are maximally collapsed). Our configuration cases are analog to those previously defined for primal methods such as MC (Figure 3.3). The table-driven orientation allows parallel approaches to be free from topology computations and improves cache efficiency in the volume accesses. In addition, this separates DC in two phases, the geometry specification and refinement of the vertex positioning inside the voxel. The separation suits both approaches of geometry creation in GPUs, especially Histogram Pyramids (section 5.3).

The table follows the idea that only three of the twelve edges are unique per voxel (x, y and z axis). Any group of three edges of the voxel with different axis orientations can be used for building the configuration table. We chose three edges that share one vertex (upper-right), resulting in four vertices to be analyzed per voxel (Figure 4.4). It is an even simpler table than the one of Marching Tetrahedra, because we have only to combine three quads. The vertices that are inside or outside the isosurface can be mapped

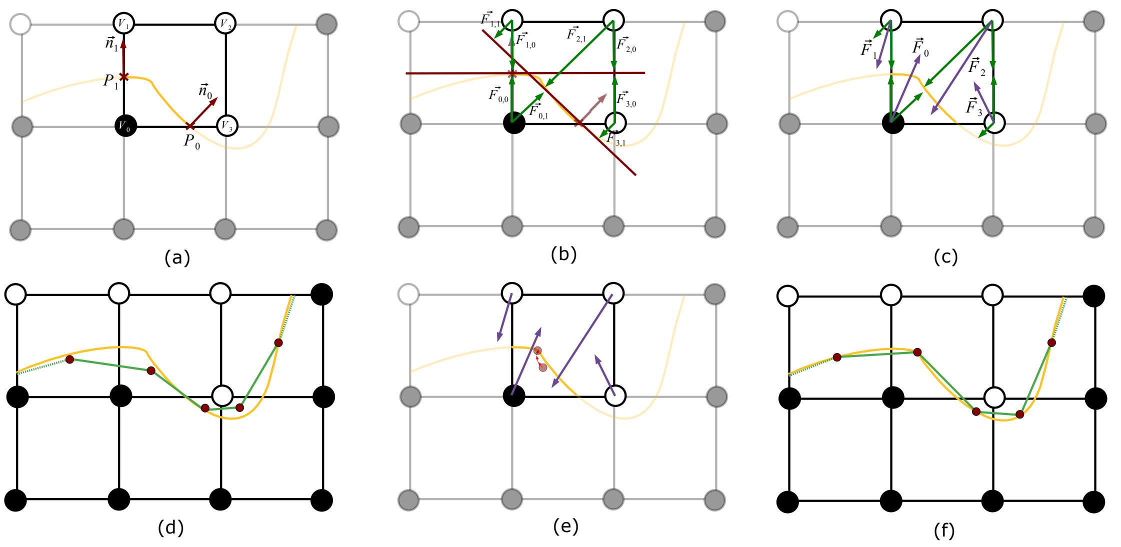


Figure 4.3: Particle-based approximation of features in Dual Contouring (2D example). (a): hermite data. (b): vectors (green arrows) from the vertices of the grid to the planes defined by the hermite data; (c): forces $\vec{F_k} = \vec{F_i} + \vec{F_{i+1}} + ...$ in the vertices of the grid; (d): initial line positioning using the mass point $\bar{C}$; (e): moving the particle towards the feature. The red arrows show the particle path. (f) Final polygonization.

to four bits. The complete table used for Dual Contouring is described in Figure 4.5.

The first element of each row is the number of vertices required for the topological case (4 for each quad used, corresponding to its 4 vertices). It is useful for determining the number of iterations of vertex emission with geometry shader and for the creation of primitives in HistoPyramids. The following elements are the neighboring voxel labels for the vertices of the quad. For instance, if the index $v_3v_2v_1v_0$ is 0010, the first vertex is located in the voxel along its z direction (V_4'), the second is in z plus x direction (V_7'), the third is in the x direction (V_3'), and the last one is the current voxel (V_0').

4.3 Summary

The first dual method was Surface Nets (GIBSON, 1998). The Surface Nets algorithm is based on the *Cuberille* sampling technique, which places vertices at the center of each active cell and connects them to vertices in adjacent active cells. The resulting mesh, as a dual of MC base mesh (in the absence of self-intersections), does not have as many badly-shaped triangles as MC. The quality of the mesh is further improved with a post-processing smoothing step, which applies a Laplacian smoothing, while constraining each vertex to the active cells.

The Dual Contouring algorithm (JU et al., 2002) combines the sampling technique of Surface Nets with the feature sensitive approach of Extended Marching Cubes (KOBBELT et al., 2001), which results in a fast and accurate polygonizer. Each vertex, instead of being positioned in the center of the active cell, is pushed to the corner of the sharp features (if they exist) in the interior of the cell. This approach improves the accuracy of the sampling technique, while maintains the efficiency of the Surface Nets.

We did two significant modifications in DC: the particle-feature approximation and the table-driven geometry specification. The first is responsible for the iteration towards the

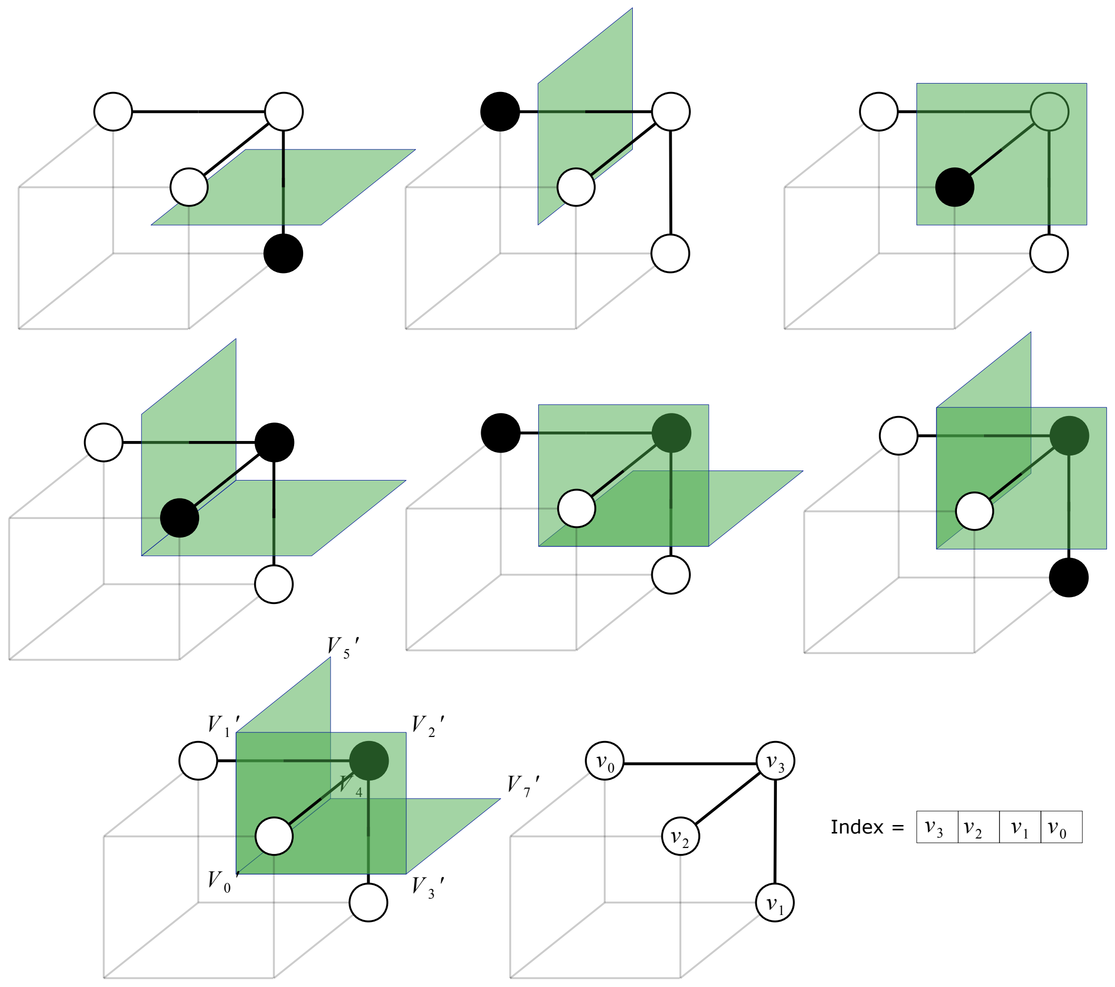


Figure 4.4: Table-driven approach for dual methods. The highlighted edges present a four bit combination $i = v_3v_2v_1v_0$ used to identify the topology of the quad by one of the predefined cases. The table is similar to the one in the MC, but has much simpler cases. For example, 1101 means the y axis is active (the vertex represented by the bit v_2 is outside the isosurface), which is, by complement, the upper-left case of the figure. It is used to identify the relative position of the vertices from the quad.

feature in a pre-established number of steps. The latter is oriented for topology discovery in a similar fashion to MC. Both modifications target a parallel implementation.

index 0000: $\sum = 0$ vertices
index 0001: $\sum = 4$ vertices, V_4', V_0', V_1', V_5'
index 0010: $\sum = 4$ vertices, V_4', V_7', V_3', V_0'
index 0011: $\sum = 8$ vertices, V_4', V_0', V_1', V_5' & V_4', V_7', V_3', V_0'
index 0100: $\sum = 4$ vertices, V_4', V_0', V_3', V_2'
index 0101: $\sum = 8$ vertices, V_4', V_0', V_1', V_5' & V_0', V_3', V_2', V_1'
index 0110: $\sum = 8$ vertices, V_4', V_7', V_3', V_0' & V_0', V_3', V_2', V_1'
index 0111: $\sum = 12$ vertices, V_4', V_0', V_1', V_5' & V_4', V_7', V_3', V_0' & V_0', V_3', V_2', V_1'
Complement from cases 0 to 7:
index 1000: $\sum = 12$ vertices, V_4', V_0', V_1', V_5' & V_4', V_7', V_3', V_0' & V_0', V_3', V_2', V_1'
index 1001: $\sum = 8$ vertices, V_4', V_7', V_3', V_0' & V_0', V_3', V_2', V_1'
index 1010: $\sum = 8$ vertices, V_4', V_0', V_1', V_5' & V_0', V_3', V_2', V_1'
index 1011: $\sum = 4$ vertices, V_4', V_0', V_3', V_2'
index 1100: $\sum = 8$ vertices, V_4', V_0', V_1', V_5' & V_4', V_7', V_3', V_0'
index 1101: $\sum = 4$ vertices, V_4', V_7', V_3', V_0'
index 1110: $\sum = 4$ vertices, V_4', V_0', V_1', V_5'
index 1111: $\sum = 0$ vertices

Figure 4.5: Indices for generating DC quads. The index is used to identify in which voxel the dual vertex V_X' is positioned. The dual vertices numeration order is described in figure 4.4.

5 GPU POLYGONIZATION

This chapter describes technology and algorithms applied in the GPU polygonization of isosurfaces. It presents implementation details of the previously-described algorithms on GPU. This helps the straightforward development of any divide-and-conquer polygonizer of isosurfaces. Since the GPU originally operated with a single primitive in each pipeline pass and many developers are not aware of stream-expansion approaches, we outline current techniques for the creation of geometry on the GPU.

Programmable GPUs first appeared on architectures such as the *PixelFlow* (EYLES et al., 1997), which was later introduced in commodity hardware. Early industry models provided a very limited set of operations. General-purpose computing and polygonization of isosurfaces were not originally designed for GPUs. The developer was restricted to the creation of small rendering programs.

The early programmable modules allowed custom shading on pixels (NVIDIA, 2003a) and vertices (NVIDIA, 2003b). The developer worked with basic functions for the production of visual effects on these elements. The CPU produced all geometry and the GPU worked with this geometry without any possibility of expansion. The polygonization of isosurfaces could not define triangles on-the-fly inside the GPU. Some approaches circumvented this issue with the assumption of a fixed number of output polygons per voxel. If there were remaining polygons (empty cells), then the geometry could be culled by fixed functionality (outside the viewing frustum, for example).

The MT polygonizer described by (PASCUCCI, 2004) is one of the first GPU approaches that performs a significant part of the extraction inside the graphics processor. The input for each tetrahedra is a quad, which is the configuration case with the maximum number of primitives. If the configuration of the tetrahedra results in the polygonization of a single triangle, then two points of the quad are collapsed into one. If no geometry is necessary at all, the quad points are discarded by collapsing them into a single point. In the case of a uniform grid, twenty or twenty four points are sent per voxel, which result in a bandwidth bottleneck. Moreover, collapsing points to the same position results in non-manifold surfaces.

Reck *et al* (RECK et al., 2004) used Span Space to reduce the amount of geometry sent to the GPU. Klein (KLEIN; STEGMAIER; ERTL, 2004) used vertex arrays and shifted computations to the fast Pixel Processors. Kipfer (KIPFER; WESTERMANN, 2005) further improved this implementation by letting tetrahedra share vertices. However, all the above algorithms perform tetrahedral subdivision, which is still undesirable for accuracy purposes (SNOEYINK, 2006) and for their excessive increase in number of triangles in comparison with MC.

The MC polygonizers described in (GOETZ; JUNKLEWITZ; DOMIK, 2005; JOHANSSON; CARR, 2006), with the help of Span Space, assumed that the number of primitives generated by each cell is performed on the CPU. Thus, excessive computation was left for the CPU, resulting in a bottleneck. On-the-fly MC only appeared recently with the new shader pipeline (TATARCHUK; SHOPF; DECORO, 2007).

Current GPUs have the geometry shader (GS) (BLYTHE, 2006), which allows the creation and deletion of primitives. This makes obsolete workarounds to create geometry and takes out excessive computations from CPU. MC and other subdivision-based polygonizers can be fully implemented inside the GPU, without any CPU direct interference. In contrast, an alternative data structure for compaction/expansion of primitives, called Histogram Pyramids (HP) (DYKEN et al., 2008), is an elegant workaround. It uses multiple passes of rendering to create the geometry out of a set of canonical indices (allocated as triangles in a VBO).

In this work, we evaluate how the GS and the HP impact the GPU implementations of Macet and DC. The GPU-based Macet algorithm shows edge transformations that improve the quality of the extracted mesh. The GPU-based DC shows improvement in the approximation of the isosurface. Both implementations explore GPU parallelism in detail and result in the fastest (to our knowledge) high-quality polygonizers available today.

5.1 Implementation Details

GPUs provide general-purpose functionality for the polygonization of isosurfaces. The implementation of these algorithms is very similar to the ones on the CPU. However, it is still necessary to optimize GPU implementations or even to test among different implementations to maximize the efficiency in results. We describe below how to implement the most efficient versions we achieved. Our novel GPU approaches for polygonization have the same basic procedures of current GPU-based MC, so one description suffices.

The input of all algorithms is the dataset, which is streamed from the CPU to the GPU memory as a 3D texture. Each value sampled in the 3D texture within two slices is parameterized to be filtered through trilinear interpolation. Values sampled outside can be either clamped to the border or repeated.

Each voxel is processed independently, according to the approach that creates primitives on the GPU. If geometry shading (section 5.2) is used, the CPU streams the vertices with coordinates identifying the voxel position. If Histogram Pyramids is used (section 5.3), each voxel is identified as a pixel from a quad. Each of these approaches has its own tradeoffs.

Current GPUs have *integer* support for textures and operations. They do not impact the efficiency of the implementation. It is possible to store the topology table in a 2D integer texture and perform a bitwise operation to create configuration indices such as MC $v_7v_6v_5v_4v_3v_2v_1v_0$ and DC $v_3v_2v_1v_0$. With this index, the polygon topology (and number of primitives) is consulted in the table texture. For the last step in MC, linear interpolation is a hardware available technology, easily performed along the edges defined by this topology.

The Macet algorithm is implemented as an extra module of the GPU-based MC. It takes place before the linear interpolation over the vertex, and has the same implementation as on the CPU. Thus, the use of Macet is straightforward in parallel hardware and there were no complex workarounds in its mapping to the GPU. The Dual Contouring algorithm has been specified with our table-driven approach. The table is responsible for the topology and geometry for the quads and our particle-based approach is responsible for the Quadratic Error Function minimization. It has an MC-like implementation and is similarly mapped to the GPUs.

5.2 Geometry Shader

The *geometry shader* allows the deletion or creation of a limited number of primitives after a vertex program. The polygonization of isosurfaces is possible in a single shading step, without any workarounds in the primitive creation. Each voxel that contains the isosurface is polygonized with this shading step. Voxels that are not crossed by the isosurface are discarded, *i.e*, the primitive corresponding to the voxel coming from the *vertex shader* is not passed to the *pixel shader*.

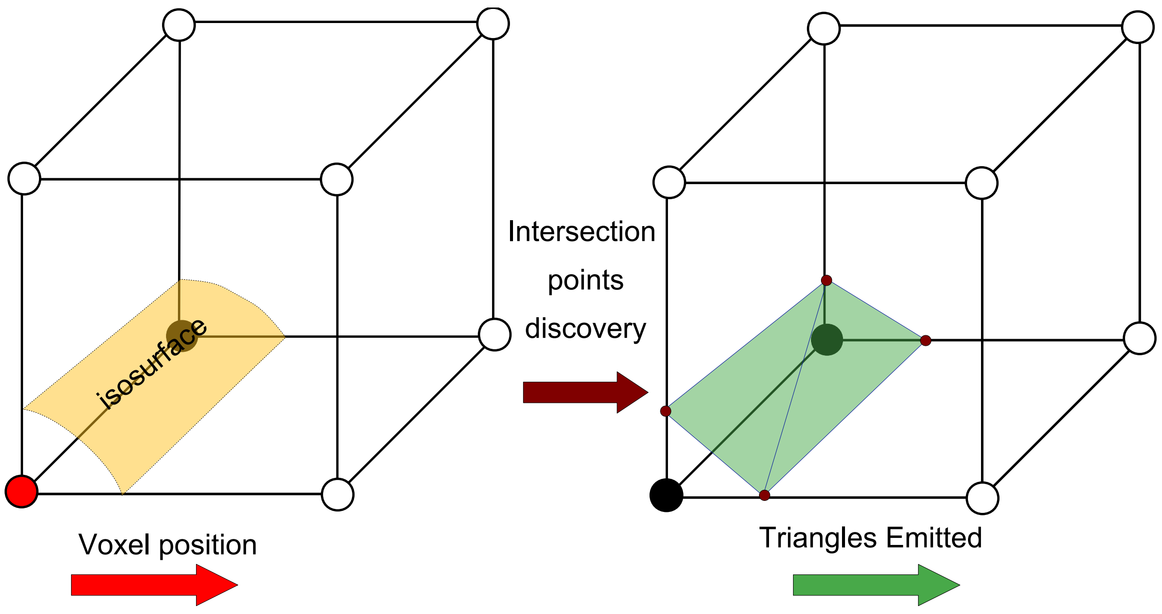


Figure 5.1: Geometry shader use for Marching Cubes algorithm. The vertex shader only passes the relative voxel position within the volume grid to the geometry shader. The point sent to the geometry shader is expanded into triangles that are emitted to the pixel shader.

When a geometry shader program is created, it is necessary to tell what primitive will come from the vertex shader (VS) and what primitives will be emitted to the pixel shader. For instance, in MC the input of the GS are points (simplest primitive) and the output are triangle strips. As MC table often relies on triangles, and not on triangle strips, the GS may end the strip after three points have been emitted.

At runtime, the CPU sends to the VS, which just passes to the GS, the voxel position, as shown on Figure 5.1. The geometry shader classifies it according to the samples in the cube vertices. Next it emits triangles consulting their connection with the enumerated triangle table texture. On the example of the figure, only two triangles are created (six points are emitted). If there is no active edge, no triangle is passed to the pixel shader at all.

Empty voxels can be avoided by Span Space on the CPU. Only cells crossed by the isosurface are sent to the GS, which just triangulates them. This accelerates even more the algorithm (results in section 6.1), but implies in preprocessing and search costs to the CPU. If the CPU has many cores, ISSUE (described in section 3.4) may be used in a multi-threaded implementation, sending multiple voxels to the GPU at the same time.

5.3 Histogram Pyramids

Histogram Pyramids (HP) is a GPU data structure that creates and discards polygons with multiple shader passes. Basically, it compacts data in a similar fashion of Scan (HORN, 2005; HARRIS; SENGUPTA; OWENS, 2007) and expands this data with a set of canonical indices. The implementation of the algorithm is not as straightforward as geometry shading, but its efficiency for polygonization of isosurfaces justifies its use.

The geometry information of the polygonizers is stored in a pyramid (stack of textures). This pyramid is constructed applying multiple pixel shader passes in quadrilaterals. Each voxel is associated with a pixel in the pyramid base. This pixel contains topology and number of necessary primitives for the polygonization. The *frame buffer object* (FBO) is used to allocate GPU memory for the textures in the pyramid. The pixel shader (last programmable module of the pipeline) streams out its elements for this buffer.

There are three steps needed to use HP: (i) the base construction, which stores geometry information; (ii) the base reduction, which eliminates uncut cells; (iii) the pyramid traversal, which expands the geometry from active cells. Each of these steps is mapped into one or more shader passes. The algorithm does not have any preprocessing cost. All of the steps are applied efficiently on current GPUs. The major problem of HP is the large GPU memory usage for the stack of textures.

5.3.1 Base Construction

The pyramid base contains geometry information of cells of the polygonizer. Its construction is the first shader pass and its storage is texture memory. A flat 3D method (HARRIS et al., 2003) is used for the mapping of the 2D pyramid base to the 3D volume, which is basically a tiling of z 2D textures into a squared texture, as can be seen in Figure 5.2. The pyramid base is drawn with a quadrilateral. Each pixel (associated with a cell) in the texture base has two distinct elements: (i) the number of vertices of the polygons, which can be triangles (three vertices each) or any other pre-specified type of primitive; (ii) the topology of the polygon, which is specified by the bitwise combination of the table of the polygonizer.

Current GPUs allow color channels to be represented by a *float* with 32 bits. Both number of primitives and topology are represented in one scalar, which reduces the amount of GPU memory required. The number of primitives is represented in the decimal part of the float and the topology in its fractional part. The fractional part representation is obtained with a division. For example, Dual Contouring has 16 different topological cases, so the division number is 16.

5.3.2 Base Reduction

The base is reduced with multiple shader passes into a single pixel. The reduction uses the information in the base about the number of primitives, while the topology is not considered. This step is analogous to the deletion of primitives on the geometry shader, where empty voxels are discarded.

The reduction is bottom-up and the texture layers follow the proportions of MipMaps. Each four pixels in the lower texture are mapped into one pixel in the upper texture, as can be seen on Figure 5.3. The function of mapping is the sum of primitives contained in the base. At the top of the pyramid there is the total amount of primitives required for the polygonization. This amount is used for the expansion step in the traversal.

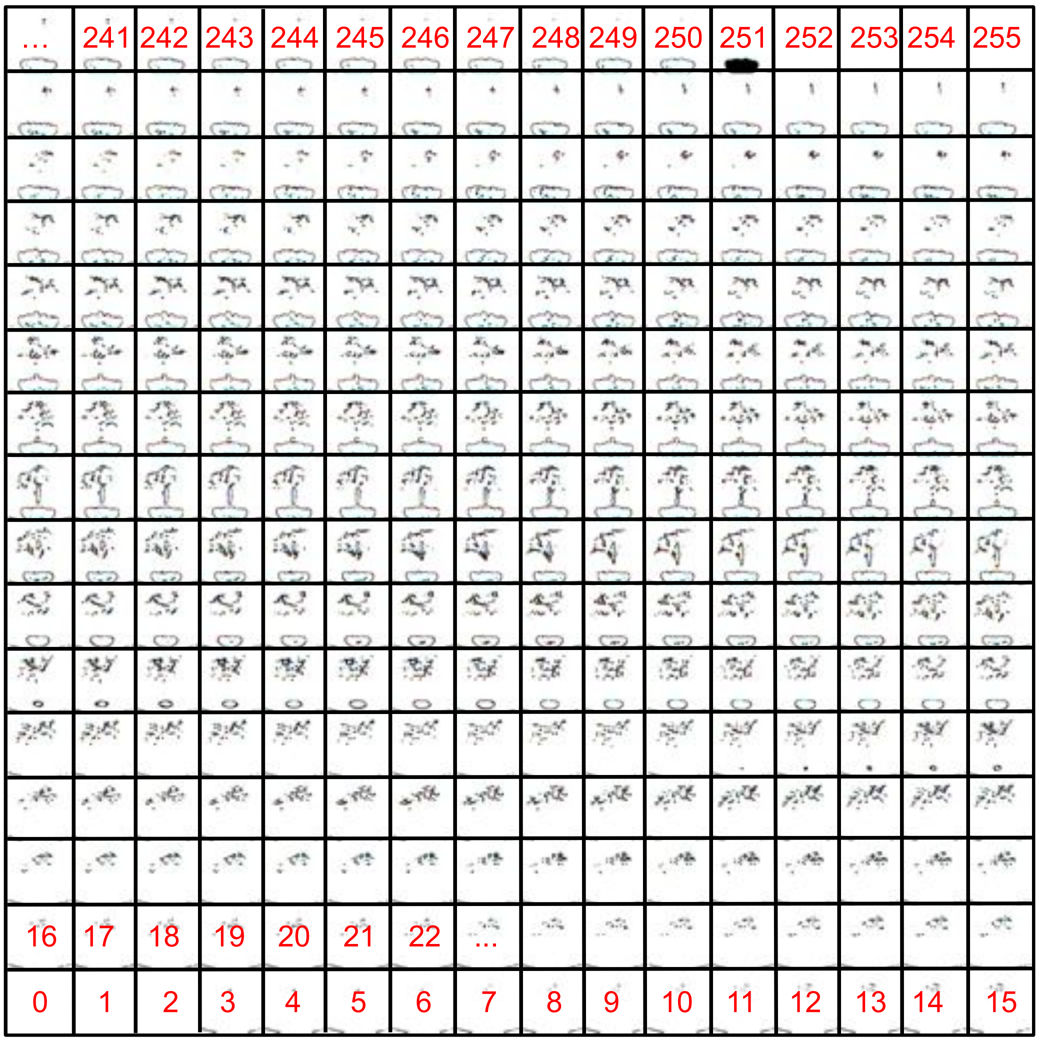


Figure 5.2: Pyramid base. The configuration case and the number of vertices of the *Bonsai* dataset are stored in this base texture. The number on the tile represents the z slice of the 3D texture.

5.3.3 Traversal

At this point, the pyramid data structure is built and the number of primitives is available on the CPU. The expansion of the primitives specified in the pyramid base happens. The single texel on the top, which contains the sum of all n vertices, is analog to a root of a tree that points at four siblings with a hashing-like function. Each texel (voxel) in the base texture is reached through unique identifiers (k), from 0 to n (sum of all vertices), which enumerate each vertex. Hence, no direct communication from the CPU is needed by the GPU, besides emitting the sum of vertices (triangles divided by three) with predefined indices.

This index k trespasses all texture levels in a top-down traversal. At each mipmap level, it is necessary to find out to which texel position $T_{x,y}$ the k' (k updated per level) points. There are four candidate siblings $T_{-1,1}$ (upper-left), $T_{1,1}$ (upper-right), $T_{-1,-1}$ (lower-left) and $T_{1,-1}$ (lower-right) per level. They are chosen based on the intervals of

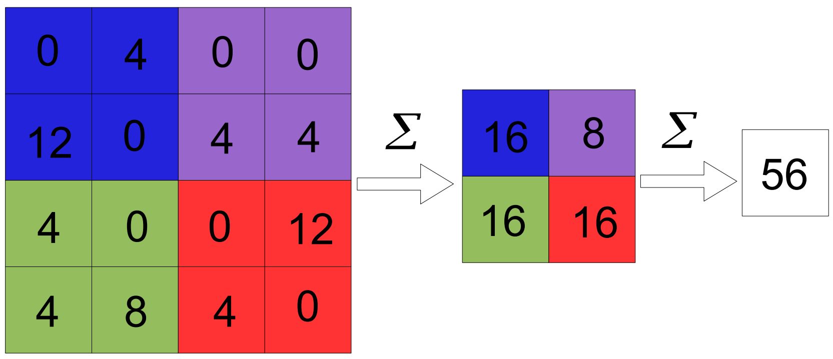


Figure 5.3: Pyramid reduction. The base has the information about the number of vertices in each voxel. It is compacted up into a single bucket.

their values ($V_{x,y}$). Let

$$\begin{aligned} x &= V_{-1,1}, \\ y &= V_{-1,1} + V_{1,1}, \\ z &= V_{-1,1} + V_{1,1} + V_{-1,-1}, \\ w &= V_{-1,1} + V_{1,1} + V_{-1,-1} + V_{1,-1} \end{aligned}$$

Therefore,

$$\begin{aligned} T_{-1,1} &= [0; x), && (5.1) \\ T_{1,1} &= [x; y), && (5.2) \\ T_{-1,-1} &= [y; z), && (5.3) \\ T_{1,-1} &= [z; w]. && (5.4) \end{aligned}$$

At the first level $k' = k$. After that, k' is updated by subtracting the first interval endpoint. At the lower level the same process is done, and it goes on until the base texture is reached. The remainder of k' represents the vertex within 0 and the sum of the vertices of the cell. This is where the expansion takes place. k' is used to access the table (patterns of Marching Cubes and Dual Contouring) that identifies where the vertex is located.

The traversal is easier to understand with a practical example (based on Figure 5.3). For instance, if $k = 26$, $V_{-1,1} = 16$, $V_{1,1} = 8$, $V_{-1,-1} = 16$ and $V_{1,-1} = 16$, then k maps to the texel $T_{-1,-1} = [y = 16+8; z = 16+8+16)$ (lower-left pixel). After that, $k' = k - y$, where $y = 16+8$. The index $k' = 2$ is applied to the same procedure to locate what texel it corresponds to in the pyramid base ($T_{-1,1} = [0; x = 4)$. The remainder is $k' = 2 - 0$, which is used to access the topology table. If this example is Dual Contouring and the topology case is 0001, then $k' = 2$ points to the third element, which is the voxel V_1' (see Figure 4.5).

5.4 Polygon Shape Results

This section outlines the basic polygon shape results from our techniques. Macet implementation did not differ from the CPU as well as its results, which were the best among our GPU-based versions. The importance of polygon shape is described in section 2.2.1.

The radii ratio average and minimum in MC is inferior to both DC and Macet (see Table 5.1). The best triangle mesh results can be seen on Macet, which has an average quality of 80 % (near equilateral edges) and its worst case is at least 200 times better than MC.

Datasets				
	Backpack	Bonsai	Engine	Stent
X x Y x Z	512^2x256	256^3	256^3	512^2x174
λ (Isovalue)	1000.5	49.5	49.5	500.5
# Tri. (MC)	2338896	671296	599024	2823756
# Tri. (DC)	2342608	673048	598920	2826208
Av.Qlty(MC)	0.700012	0.687802	0.70733	0.71089
Av.Qlty(DC)	0.766	0.75725	0.77274	0.77473
Av.Qlty(Macet)	0.81369	0.82694	0.81674	0.80432
Min.Qlty(MC)	1.525e-5	1.373e-3	4.232e-4	1.709e-5
Min.Qlty(DC)	1.785e-5	1.138e-4	6.55e-3	1.502e-3
Min.Qlty(Macet)	0.44475	0.47395	0.51064	0.45069

Table 5.1: Volume and isosurface extraction information. The average and minimum triangle quality is given by the radii ratio.

One downside of the GPU implementation of Macet in (DIETRICH et al., 2008) and (DIETRICH et al., 2009) is that the combination of both gradient and tangent transform triangle meshes has complex topology requirements (accessing triangles in neighbor cells). However, considering that the combination can be done in milliseconds by the CPU using both GPU meshes, this is not a problem. Moreover, the bottleneck of these Macet versions is the mesh extraction and not their combination. The single step version that is shown in (DIETRICH et al., 2009) does not need any combination and was implemented on our GPUs.

5.5 Summary

This chapter revisited two proposals for geometry creation on GPUs. It described how to use these approaches for the polygonization of isosurfaces. The first relied on implementing the entire algorithm inside the geometry shader (GS) feature. Such feature represents a programmable stage added into the graphics pipeline to allow limited control on the creation and deletion of vertices (and consequently triangles). Acceleration structures such as Span Space (SS) (SHEN et al., 1996) can also be combined with the GS for improved performance. A second approach to the problem is to use a multi-pass solution using the pixel shader, called HistoPyramids (HP) (DYKEN et al., 2008). Results using this method are even better than using the GS, but it also has shortcomings, since it requires large GPU memory allocation and may limit the size of input volumes. We also showed the polygon shape results from the extraction of isosurfaces with GPU-based Macet.

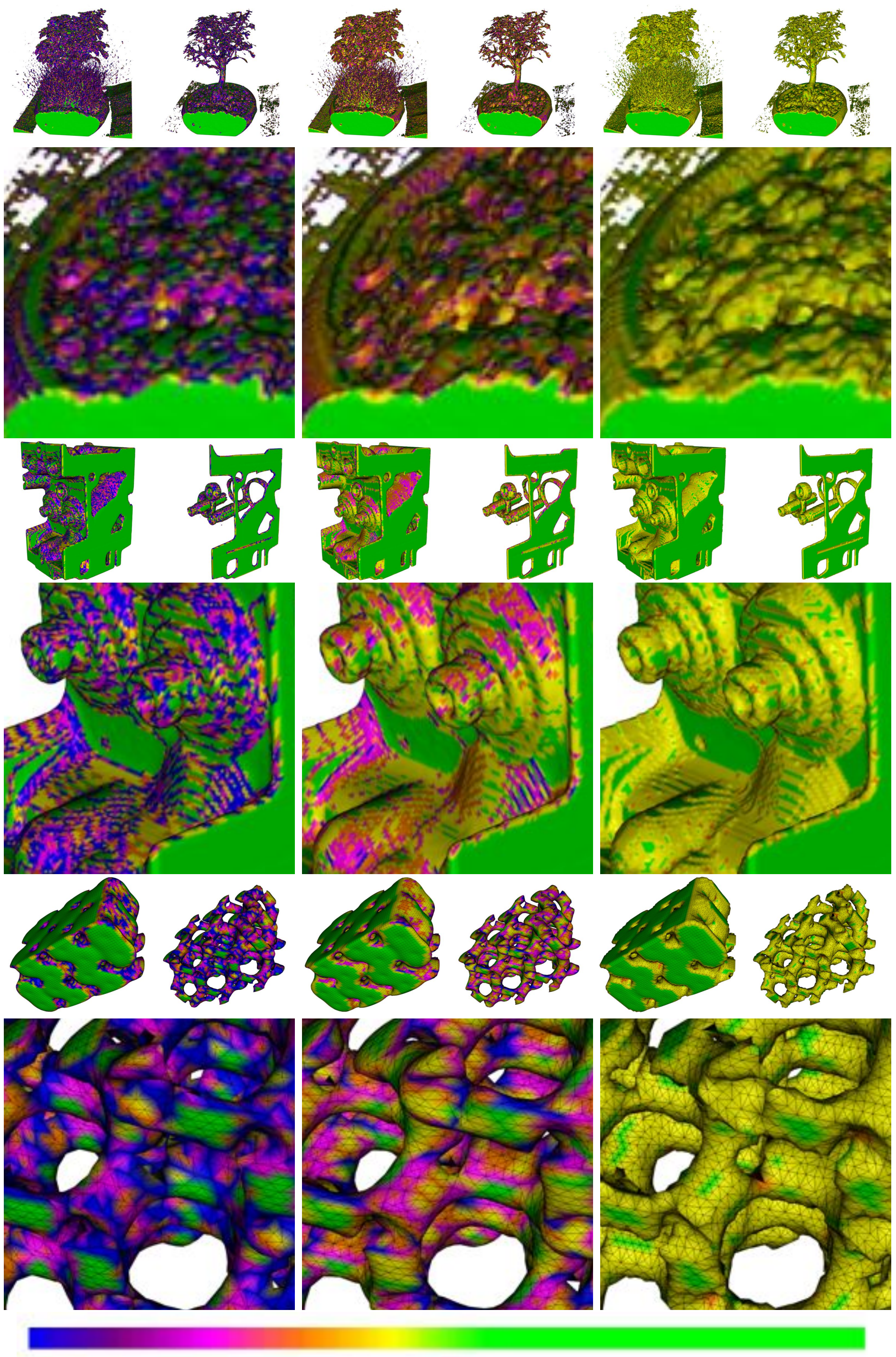


Figure 5.4: Polygon shape Results. Interactive navigation through isosurfaces results and radii ratio coloring (green = good, blue = bad) triangle vertices. Bonsai (row 1-2), Engine (row 3-4) and Silicium (row 5-6) datasets. Marching Cubes (column 1), Dual Contouring (column 2) and Macet (column 3) polygonizations with two different isovalues.

6 EFFICIENCY ANALYSIS

This chapter presents, in two sections, the results from the exploration of GPU capabilities in the polygonization of isosurfaces. Section 6.1 shows the efficiency of our two novel techniques previously described in comparison to the fastest version of MC on the GPU. It also shows how fast the GPU can extract isosurfaces in comparison to the CPU. Section 6.2 profiles the different GPU implementations of MC in order to find the graphics hardware bottlenecks and proposes improvements in its architecture.

We implemented our methods in GLSL (OpenGL® Shading Language) running on two different configurations, which we called A and B. Configuration A is a GeForce[1] 8800 GTX with 768 MB of VRAM, an AMD Athlon[2] 64 Processor of 2.2 GHz and 2 GB of RAM DDR1, running on a motherboard ASUS A8N SLI Premium. Configuration B is a GeForce 9600 GT with 512 MB of VRAM, an Intel Core 2 Quad 2.4 GHz and 4 GB of RAM DDR2, running on a motherboard ASUS P5N E.

The reason we chose these configurations is their different computational power. Configuration A has a more powerful GPU, but a much simpler CPU, less main memory and an older motherboard. Thus, we can find out in what scenario a very powerful GPU by itself suffices for real-time visualizations. Moreover, the analysis of these different configurations shows the impact of each low-cost hardware element in computationally demanding algorithms such as isosurface polygonization.

The volume data used for our results were *Aneurysm*, *Backpack*, *Stent*, *Bonsai*, *Engine* and *Silicium* (VOLVIS, 2008). None of them is time varying and all are online and available for download. Size and dimension of the datasets can be seen on Table 6.1. The *Aneurysm* was reassembled (trilinear filtering) into a larger and more detailed dataset, but still small enough for the main memory of a 32-bit operational system. We did not consider *out-of-core* solutions in this work (terabyte order datasets), due to high I/O demand on the hard disk, which is prohibitive for low-cost hardware.

	Aneurysm	Backpack	Stent	Bonsai	Engine	Silicium
X x Y x Z	1024^2x512	512^2x373	512^2x174	256^3	256^2x128	64^3x128
Sample precision	unsigned short	float	float	float	float	float
Memory (MB)	1024	373	174	64	32	2

Table 6.1: Volume dimensions and memory size.

The *vertex buffer object* (VBO) is used to decrease the bandwidth requirements on CPU to GPU communication. The use of this technology implies in more memory consumption, but it may accelerate even further the isosurface extraction. If there are too

[1]Copyright ©2007 nVidia Corporation
[2]Copyright ©2007 Advanced Micro Devices, Inc.

many primitives (points in GS and triangles in HP) streamed into the GPU, then the trade-off for memory may be useful.

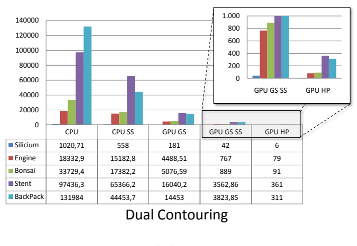
The *transform feedback* (TF) extension (GLSL v1.20) is used to save the geometry on the GPU as a Vertex Buffer Object. The inclusion of this technology brought two advantages: polygonization of the isosurface only once per isolevel and, more importantly, allowing the CPU access to normal and vertex coordinates (streaming out the mesh to the CPU). If the purpose of the polygonization is the pure visualization of isosurfaces, then the inclusion of TF might be unnecessary, especially if the extraction takes less than one second to happen.

6.1 Comparison of Techniques

We compare the different polygonizers running on the most efficient versions of geometry shader (GS) and Histogram Pyramids (HP) of our implementations. The claim here is that, with little overhead, the GPU can polygonize more sophisticated techniques than pure MC and result in better polygon meshes. We already enumerated why having a high quality with regard to distance to the isosurface and polygon shape is fundamental for many applications. Besides, we also show that the GPU versions are far more efficient than the CPU ones.

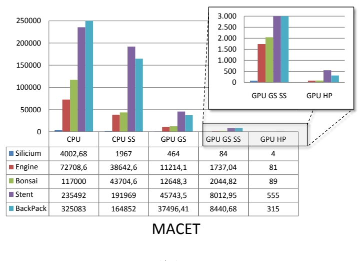
Our intent with this section is to prove that our novel technique obtained similar results than MC in its efficient versions. To achieve this, it is necessary to run different GPU isosurface extractors, with one hardware configuration. Configuration A was used for these tests.

We only used volumes that fit on the GPU memory and that could also be used along the memory required by all techniques. HistoPyramids, which requires the most memory, consumes much of the GPU memory for the pyramid storage. Therefore, the *Backpack* dataset had some z slices taken out and simplified into 256 slices. We also used smaller datasets such as *bonsai*, *engine* and *stent*.



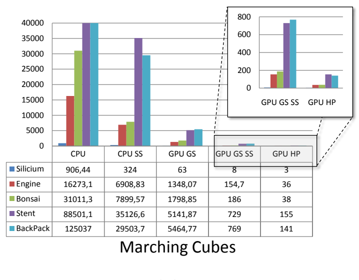
	CPU	CPU SS	GPU GS	GPU GS SS	GPU HP
🟦 Silicium	1020,71	558	181	42	6
🟥 Engine	18332,9	15182,8	4488,51	767	79
🟩 Bonsai	33729,4	17382,2	5076,59	889	91
🟪 Stent	97436,3	65366,2	16040,2	3562,86	361
🟦 BackPack	131984	44453,7	14453	3823,85	311

(a)



	CPU	CPU SS	GPU GS	GPU GS SS	GPU HP
🟦 Silicium	4002,68	1967	464	84	4
🟥 Engine	72708,6	38642,6	11214,1	1737,04	81
🟩 Bonsai	117000	43704,6	12648,3	2044,82	89
🟪 Stent	235492	191969	45743,5	8012,95	555
🟦 BackPack	325083	164852	37496,41	8440,68	315

(b)



	CPU	CPU SS	GPU GS	GPU GS SS	GPU HP
🟦 Silicium	906,44	324	63	8	3
🟥 Engine	16273,1	6908,83	1348,07	154,7	36
🟩 Bonsai	31011,3	7899,57	1798,85	186	38
🟪 Stent	88501,1	35126,6	5141,87	729	155
🟦 BackPack	125037	29503,7	5464,77	769	141

(c)

Figure 6.1: Extraction times (in miliseconds) from our GPU-based Dual Contouring and Macet compared to Marching Cubes.

Figure 6.1 shows that even with the help of Span Space acceleration, no CPU-based polygonizers reach interactive frame-rates in any dataset. The CPU hardware is not adapted (yet) to take advantage of the independence between active cells, and the inherent parallelism of domain subdivision methods. Results obtained on the GPU, on the other hand, expose its hardware capabilities. The HistoPyramids-based implementation is significantly faster than CPU versions (up to 1500 times faster), even with relatively large datasets.

DC and Macet are computationally more expensive than MC. They take approximately twice the time MC took for isosurface polygonization. These results were expected due to the higher quality polygonization. The fact that MC polygonizations are computed in milliseconds on the GPU justifies the overhead for the polygonization of better quality isosurfaces.

It is interesting to observe that the bottleneck of HistoPyramids, when the volume fits the GPU memory, is related to the number of triangles and not to the volume size, similarly to Span Space. This does not imply that the implementation suffers with a large number of triangles, as shown on Table 6.2. Sparse isosurfaces, conversely, result in very fast polygonizations even with large datasets.

	Backpack	Bonsai	Engine	Stent
λ (Isovalue)	1000.5	49.5	49.5	500.5
# Triangles (MC)	2 338 896	671 296	599 024	2 823 756
# Triangles (DC)	2 342 608	673 048	598 920	2 826 208

Table 6.2: Number of triangles generated in Marching Cubes and in Dual Contouring. Each quad of DC is separate in two triangles.

One important disadvantage of Histogram Pyramids is the excessive memory consumption. It is a challenging task to maintain large datasets, such as the *backpack*, in GPU memory along with the data structure and other GPU structures (vertex buffer objects to save the polygons and to traverse the pyramid, for instance). Another drawback is the square texture requirement, which increases quadratically and often needs padding. For example, if the volume data has dimensions of 256^3, it needs a 4096^2 base texture for the pyramid, which fits perfectly. On the other hand, if one extra volume slice is added ($256^3 + 256^2$), the base increases to 8192^2, wasting lots of memory with padding. Since the base must be squared it is not possible to add a single tile.

6.2 Hardware Profiling

This section tries to shed some light in the performance of basic architectural modules of the GPU in the polygonization of isosurfaces. Our objective is to show that there are issues in the graphics pipeline related to this work. For instance, how can a data structure such as HP be more efficient than GS, which is a hardware implemented technology for stream expansion. Another important problem is the pipeline constraints that may affect the polygonization of isosurfaces. These issues are important and we obtained detailed GPU information with the help of profiling tools.

We collected results from 10 different implementations of MC with both A and B hardware configurations:

1. HP - VBO - TF, HistoPyramids with Vertex Buffer Object and Transform Feedback.

2. HP - VBO, HistoPyramids with Vertex Buffer Object.

3. HP - TF, HistoPyramids with Transform Feedback.

4. HP, just HistoPyramids.

5. GS - VBO - TF, Geometry Shader with Vertex Buffer Object and Transform Feedback.

6. GS - SS - TF, Geometry Shader with Span Space and Transform Feedback.
7. GS - VBO, Geometry Shader with Vertex Buffer Object.
8. GS - SS, Geometry Shader with Span Space.
9. GS - TF, Geometry Shader with Transform Feedback.
10. GS, just Geometry Shader.

The volume data used in these tests were *silicium*, *bonsai*, *backpack* and *aneurysm*. As we wanted a scenario for navigation of isosurfaces, each dataset had twenty one different isosurfaces polygonized (Table 6.3). Massive results are important for reliability since a great amount of variables is analyzed. Moreover, other applications such as the operating system memory swap occasionally affect results, considering its I/O demanding costs. Another important consideration is that these datasets have increasing memory sizes (Table 6.1).

Step	Aneurysm		Backpack		Bonsai		Silicium	
	λ	# Triangles	λ	# Triangles	λ	# Triangles	λ	# Triangles
0	1500.5	369 054	1500.5	1 057 600	130.5	662 824	140.5	32 280
1	1450.5	397 906	1450.5	1 077 880	125.5	621 156	135.5	34 936
2	1400.5	429 622	1400.5	1 098 328	120.5	581 384	130.5	38 192
3	1350.5	466 598	1350.5	1 132 876	115.5	551 432	125.5	40 464
4	1300.5	507 856	1300.5	1 320 868	110.5	529 248	120.5	40 080
5	1250.5	554 754	1250.5	1 649 424	105.5	514 020	115.5	40 080
6	1200.5	609 302	1200.5	1 935 280	100.5	502 124	110.5	39 888
7	1150.5	671 760	1150.5	2 238 888	95.5	492 100	105.5	39 444
8	1100.5	745 768	1100.5	2 546 200	90.5	480 748	100.5	39 688
9	1050.5	831 018	1050.5	2 816 956	85.5	470 604	95.5	39 728
10	1000.5	934 210	1000.5	3 012 820	80.5	459 308	90.5	39 640
11	950.5	1 056 686	950.5	3 229 920	75.5	448 828	85.5	39 896
12	900.5	1 206 806	900.5	3 436 416	70.5	440 116	80.5	40 000
13	850.5	1 391 984	850.5	3 611 284	65.5	435 812	75.5	39 768
14	800.5	1 626 098	800.5	3 763 772	60.5	446 184	70.5	39 664
15	750.5	1 938 236	750.5	3 900 828	55.5	491 404	65.5	39 808
16	700.5	2 386 450	700.5	4 060 512	50.5	622 280	60.5	39 816
17	650.5	3 089 764	650.5	4 305 840	45.5	832 772	55.5	39 688
18	600.5	4 378 990	600.5	4 615 804	40.5	1 020 516	50.5	39 056
19	550.5	7 593 614	550.5	4 932 648	35.5	1 041 280	45.5	38 944
20	500.5	17 353 656	500.5	5 249 772	30.5	1 212 724	40.5	38 544

Table 6.3: Number of triangles and isovalues in dataset navigation. Except for *silicium*, larger number of triangles is extracted in the last steps of the navigation.

6.2.1 Memory Management

There were two datasets in our results that required different processing. The *backpack* occupies much of the GPU memory of our chosen configurations. This is even worse in the case of the *aneurysm*, because the dataset alone does not fit the memory of neither A nor B hardware configurations. Combining this with the fact that there are elements such as TF, VBO and pyramid of HP that also consume significant GPU memory, management becomes an important issue.

Our memory management scheme is simple. The volume is composed of z slices, which can be separated into partitions. The size of the partitions is based on the memory requirement of the volume size, combined with other elements of the GPU that also consume memory. These additional elements are often proportional to the size of the volume. For example, in HP, the pyramid base is calculated as a flat 3D volume, which is based on the size of the dataset. Thus, we maximized the GPU memory usage considering this type of representation.

Memory use in *Transform Feedback* (TF) has a direct relation to the volume in the management technique. TF allocates GPU memory with a static size specified by the developer. We have to know beforehand how many triangles the isosurface needs for its extraction. With HP, this number is available after the stream reduction step. However, it is too costly to allocate buffers with different sizes for each isovalue. Thus, we use the same memory assumption for both HP and GS. The polygonization would not be more than 25% of the triangles of the total number of cells in the volume for large datasets. This assumption is reasonable since the isosurfaces are sparse in the volume, even in situations where the data volume has noise. Besides, if this memory allocation is not enough and the number of triangles exceeds the estimate percentage, more memory is allocated. A simple query in the number of triangles written by the polygonizer would indicate the amount necessary.

The *Vertex Buffer Object* (VBO), in the HP-VBO approach, has the same number of triangles of TF. This is due to the fact that the triangles of the VBO are the triangles used for the pyramid traversal, which are saved in TF. On the other hand, in the GS-VBO approach, the VBO represents input points which are expanded into triangles. The number of necessary points for the VBO is proportional to the number of volume cells.

Memory management is important for an automatic and interactive polygonization of isosurfaces in larger datasets. The division of the dataset into multiple partitions is easily implemented, without any additional preprocessing. As one of our goals is to fully understand the context of GPU polygonization of isosurfaces, the memory management also helps in the understanding of scalability in GPU-based approaches. In addition, it allows the polygonizer to allocate appropriate sizes for TF, which requires much GPU memory for the mesh storage.

6.2.2 Hardware and Software Benchmarking

The profiling tool we used is offered by nVidia (NVIDIA, 2008a). It is called *performance toolkit* (*PerfKit*) and works for both DirectX and OpenGL. Basically, it provides access to software and hardware counters, *i.e.*, driver and GPU modules information. *PerfKit* does not work with a standard driver. It requires an instrumented version. We ensured that the driver instrumentation did not impact our polygonization efficiency. There were no significant differences in the results with the instrumentation enabled or disabled.

The hardware counters include a wide range of programmable and fixed GPU functionality (Figure 6.2). The library function that retrieves these counters can be directly integrated to the implementation. It is possible to retrieve hardware counters at any frame. *PerfKit* also provides driver information. It helps the investigation of external factors to the graphics hardware, such as the driver wait for the CPU. The counters cannot be directly extracted in implementation as hardware counters were. There is an intermediate operational system interface (*Performance Data Helper*). We used the *Microsoft*© *Management Console* to retrieve the information about these software counters.

The software and hardware counters sampled are the following (except for the last two, which are not provided by *PerfKit*):

1. **GPU busy**: percentage of time the graphics processor unit is globally busy. It is

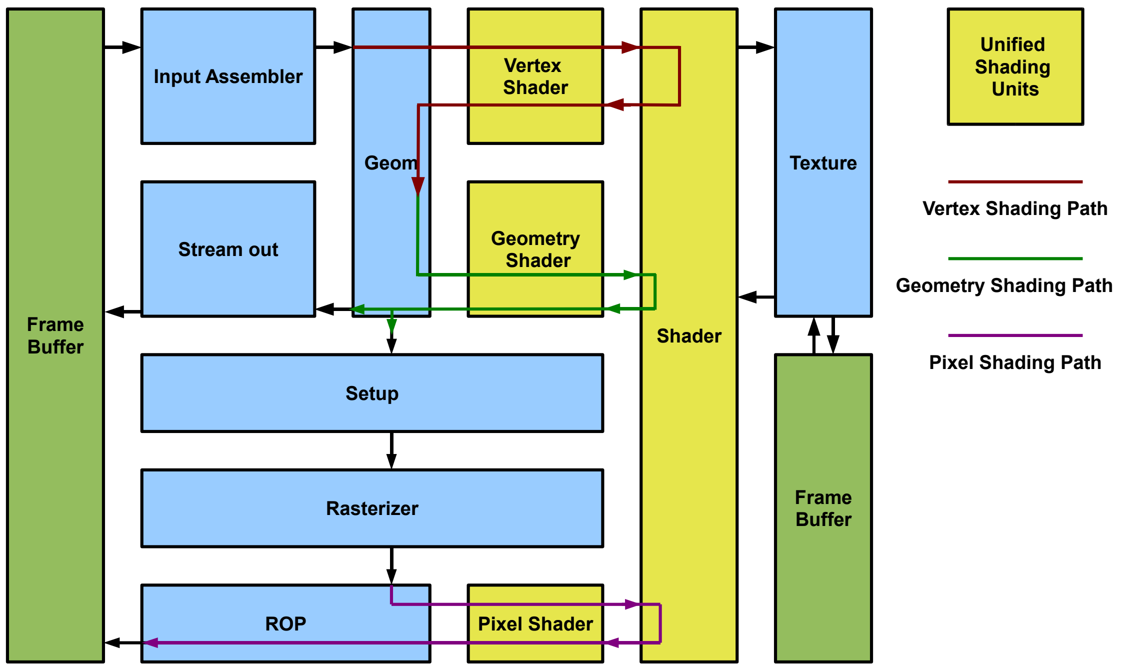


Figure 6.2: nVidia G80 architecture. This diagram shows the graphics pipeline and what modules are part of VS, GS and PS rendering steps. This architecture image is specified in (NVIDIA, 2008a).

important to the investigation in case that the CPU is overloaded and the GPU is idle.

2. **Input Assembler Busy**: percentage of time the GPU is assembling client buffers into primitives.
3. **VS busy**: percentage of time the vertex shader unit is busy.
4. **GS busy**: percentage of time the geometry shader unit is busy.
5. **PS busy**: percentage of time the pixel shader unit is busy.
6. **Shader busy**: percentage of time of shader operations.
7. **Rop busy**: percentage of time in rasterization.
8. **Texture busy**: percentage of time in texture sampling.
9. **Stream Out busy**: percentage of time the TF is taking.
10. **Geom busy**: percentage of time Geom is busy.
11. **AGP/PCI-E usage (MB)**: PCI-Express memory usage (MB).
12. **Driver waiting**: percentage of frame time the driver waits for CPU.
13. **Video Memory (MB)**: Graphics hardware memory usage (MB).

14. **CPU time**[3]: percentage of time the application uses the CPU.

15. **Extraction time**: milliseconds taken for the isosurface extraction.

The combination of these counters with the navigation through the volume data resulted in more than 18000 values. The interpretation and presentation of relevant results are addressed in the next section.

6.2.3 Bottlenecks

This section has the purpose of presenting interesting results of different combinations of *PerfKit* counters, MC extraction approaches, hardware configurations and datasets. Important facts of GPU polygonization were filtered by the analysis of some particular group of variables among a massive number of combinations. For example, results showed that bus bandwidth was never an issue. The only extraction approach that significantly consumed the bandwidth was GS without VBO in the extraction of large datasets. The *AGP/PCI-E usage* counter showed that there was a constant exchange of information between CPU and GPU. Given that CPU batches of geometry carried too much information, the *Input Assembler* was also significantly busy (about 20%). Therefore, the fact that bus bandwidth is massively loaded with data appears at this specific scenario found with these specific variables.

One clear pattern in our results is related to the size of the dataset combined with the geometry creation approach. Volume data that required our memory management technique impacted external factors of the GPU, especially using HP. The datasets that fit on GPU memory along other data structures showed an overall different behavior, having GPU modules as bottlenecks. Exceptions to this pattern were GS approaches that did not use SS. They were not affected by the memory management.

We first consider volume data that required memory management. When the volume did not fit GPU memory, the CPU was a barrier for HP and GS-SS in both configurations A and B. This issue becomes evident in configuration A, since the CPU is relatively obsolete. The navigation through isovalues in *backpack* used the GPU less than 50 % of each frame time, as can be seen in Figure 6.3 (the number of partitions in the memory management is described in Table 6.4). Other counters such as the *CPU time* and the *Driver Waiting*, which does not appear in the figure, further confirmed that the GPU was left idle. These counters showed that the CPU was 100 % busy and the driver waiting for the GPU was minimal. In addition, the isosurface navigation in *aneurysm*, another large dataset, had a similar behavior.

Excessive data handling in the CPU is responsible for a large part of the GPU idleness. The CPU has to stream parts of the volume from RAM to GPU memory at every frame, multiple times. This is too costly even for a core 2 quad such as in configuration B. Notice that the bottleneck does not reside in the PCI-Express bus, but in the memory handling by the CPU. In HP, there are excessive CPU shader state changes for the multiple rendering passes, which also overload the CPU. In GS-SS, the search and streaming of active cells is expensive for the CPU, even if the query is optimal. This problem is similar to those in older GPU polygonization techniques, such as the ones described by Pascucci and Johansson *et al.*

One possible solution for the CPU overload is the increase in GPU memory. As memory continues to get cheaper, next generation GPUs could boost their amount of

[3]Data provided by the operational system.

Approach	# partitions	GPU Memory use (MB)
HP-VBO-TF	6	485.33
HP-VBO	3	565.33
HP-TF	6	437.33
HP	2	597.33
GS-VBO-TF	5	680
GS-SS-TF	4	660
GS-VBO	3	630
GS-SS	1	373
GS-TF	4	660
GS	1	373

Table 6.4: Partitioning in *backpack* for configuration A. The number of partitions is maximized according to other data structures in GPU memory.

GPU memory and minimize CPU load (and avoid memory management). However, if datasets also increase in size at the same proportions, increasing GPU memory may not be a solution for all cases. In addition, GS-SS still would suffer from the search of active cells on the CPU. Another alternative may be building multithreaded solutions for many core CPUs. If a CPU has considerable computational power it might at least be able to transfer the volume fast enough for the GPU.

Now we discuss volume data that does not require memory management. When both dataset and extraction approaches fit on the GPU memory, GPU cores were busy most of the frame time. HP and GS-SS have their extraction time affected by the number of triangles of the surface for both configurations A and B. *Extraction time* counter in *bonsai* dataset shows this proportionality (Figure 6.4). This happens in HP because texture accesses in the pyramid base construction are not as expensive as in the pyramid traversal and the edge linear interpolation. Even though there are not as many output primitives as input cells, the handling of geometry is still more expensive. *VS Busy* counter shows that the *vertex shader* becomes busier while the *PS Busy* counter does the opposite. In GS-SS the GPU module most busy is the geometry shading step.

One obvious way of improving the situation in which the GPU is fully utilized is accelerating even more the GPU cores and texture memory fetches. However, as the extraction time is already minimal, the acceleration may not be necessary. It would still be more interesting to increase GPU memory so that larger datasets could be polygonized without any memory management.

The polygonization approaches that were not affected by the memory management used GS without SS. The problem of these approaches is the geometry shading module, which is extremely expensive for current GPUs. The bottleneck has a direct relation to the number of output primitives **per voxel**. MC is not a heavy output algorithm, as tessellation is, but already exceeds the maximum threshold that maintains GPU peak performance (20 scalar attributes) by far (NVIDIA, 2008b). MC has a triangulation case with five primitives, combined with normal information, resulting in 75 scalar attributes. Thus, the geometry shading needs to be improved for next generation GPUs.

Finally, the question of what is best choice for geometry creation can be answered. Our results show that HP is a better choice despite its multiple rendering passes and high memory demand. Even with SS algorithmic acceleration, GS is slower. The GS bottleneck is severe. For example, looking at the extraction results of *silicium* dataset (Figure 6.5), the GS step appears as a harsh constraint. The GS implementation combined with *bonsai* and configuration A resulted in GS being up to 50 times slower HP. If GS is im-

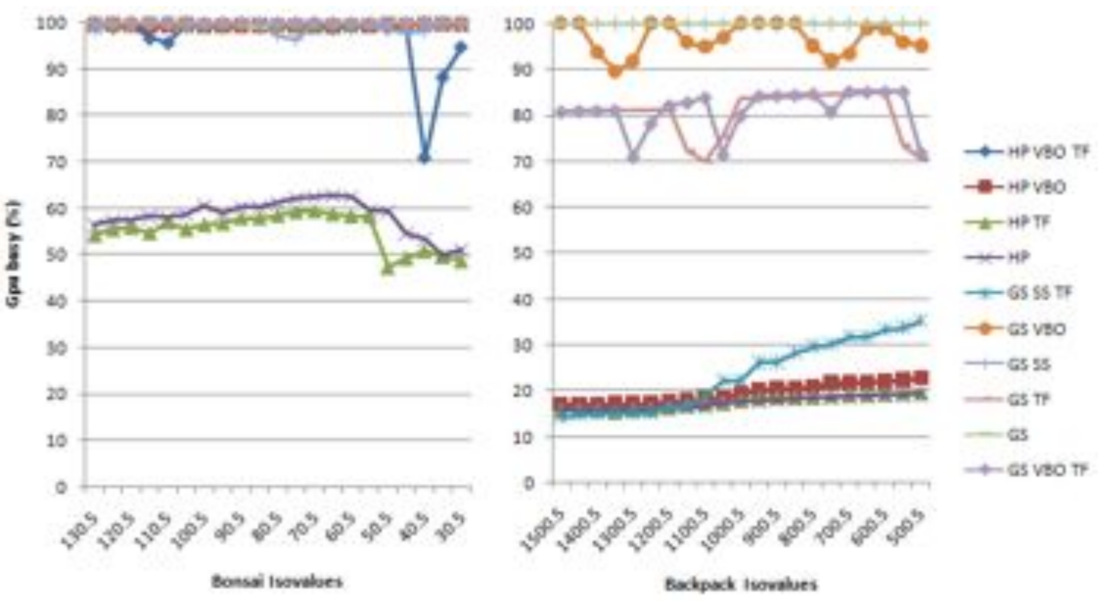


Figure 6.3: GPU busy counter in configuration A. Since *Bonsai* fits on GPU memory, the GPU was busy for most polygonization approaches, except for HP without VBO, which was affected by the wait for triangles of the CPU. In *backpack*, HP and GS-SS-TF were idle more than 50% of the time, because they required memory management. Other GS approaches offloaded the CPU because the geometry shading step is expensive.

proved and does not diminish GPU peak performance anymore, at least for a small amount of triangles, there is still one reason why one would use HP in isosurface polygonization: the access from one polygon vertex to the others. If developers allow one TF buffer to be fully and coherently accessed from within the GPU and accelerate GS, then HP has no advantage over GS.

6.3 Summary

This chapter presents the results of our novel approaches of high-quality polygonizations in the GPU and compares it to the most efficient GPU-based MC in the literature. Results show a clear performance improvement over CPU implementations and also show that our novel GPU-based approaches, Macet and DC, reached interactive results. The chapter also analyses architectural modules and issues of GPUs in the polygonization of isosurfaces. Our results indicate that GS is a very powerful shading module, but is still much slower than HP. We show where each polygonization approach has issues and propose improvements for the architecture in each of these problems. In addition, we suggest that the next generation hardware improve the efficiency of GS, since it has its performance compromised with few output primitives.

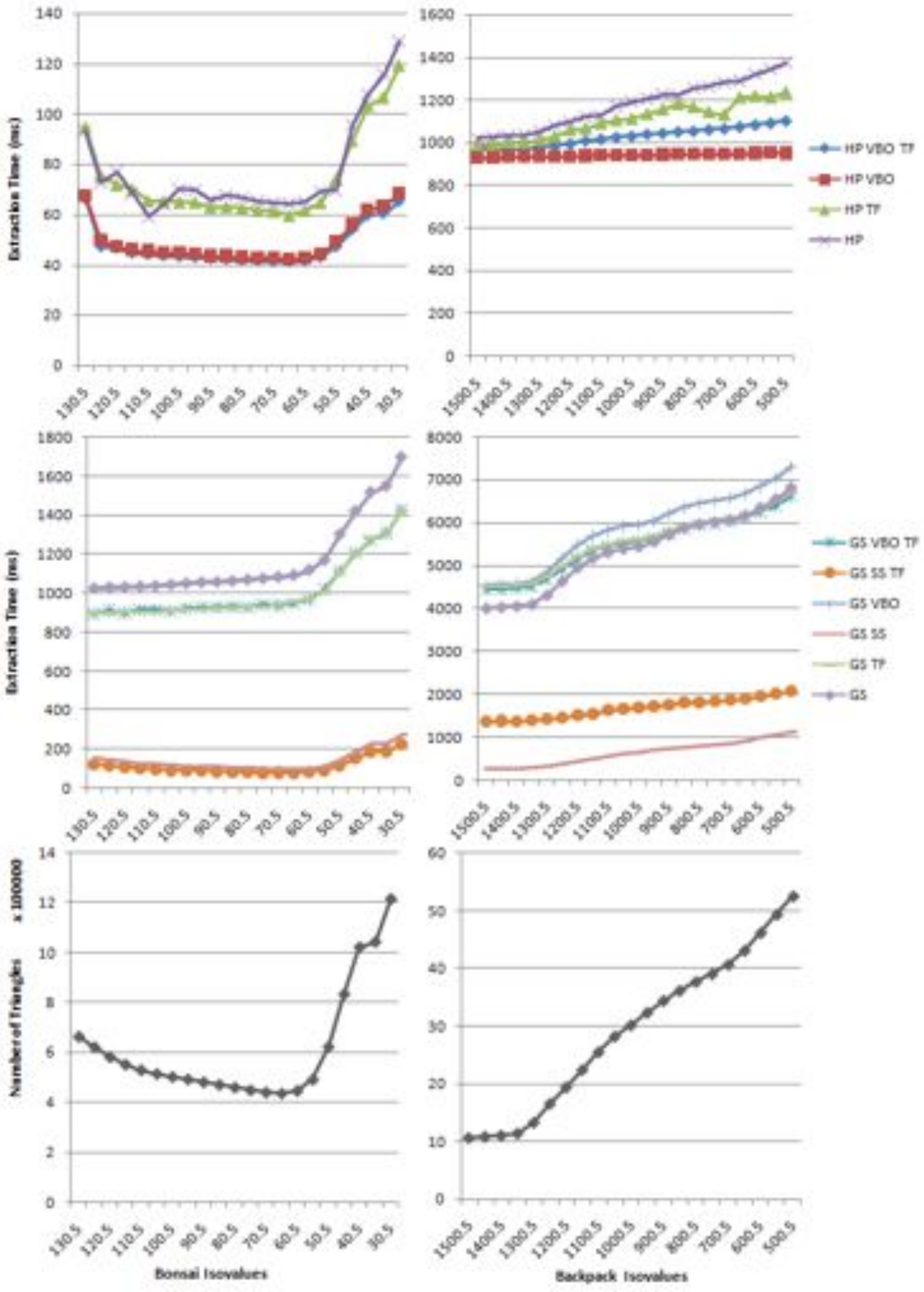


Figure 6.4: Extraction times in configuration B.

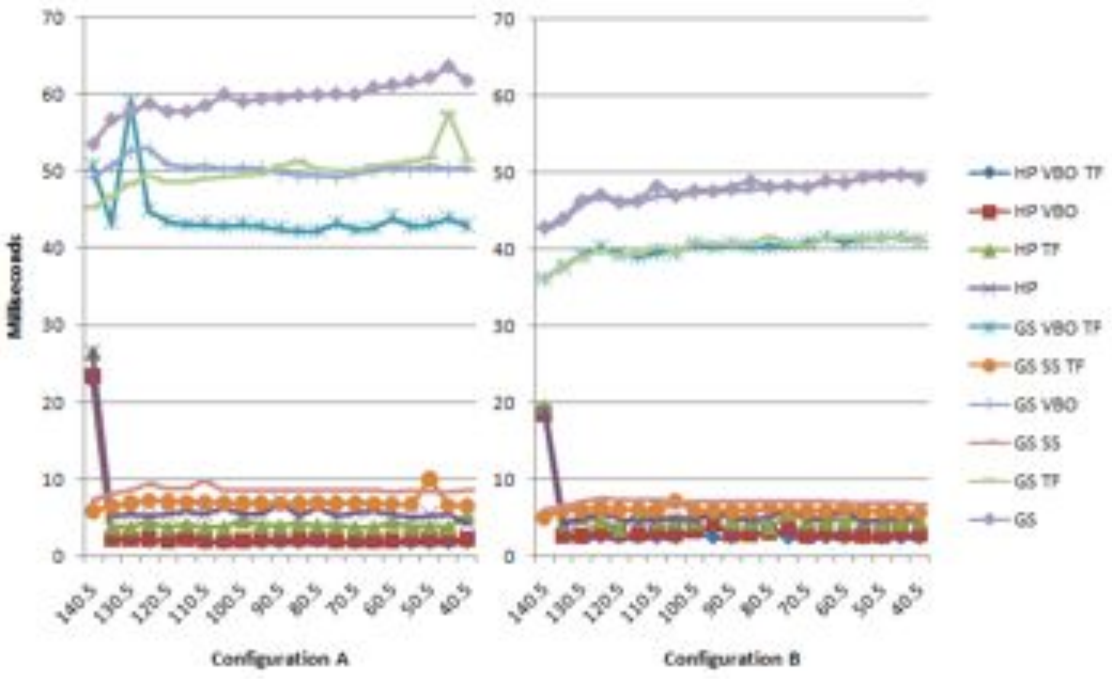


Figure 6.5: Geometry shader vs Histogram Pyramids. The 9600 GT graphics card shows improvement in the geometry shader module, but is still no match for Histogram Pyramids.

7 CONCLUSION AND FUTURE WORKS

We presented an analysis and an extension of the GPU implementations of the main isosurface polygonization algorithms. This analysis showed how the GPU can be used for high-quality polygonization of isosurfaces at interactive frame-rates. Applications that benefit from our work range from scientific visualization to simulation in multidisciplinary areas. Features such as sharp edges and corners are now possible to be visualized in isosurfaces in real-time. Tetrahedral mesh generation and finite element analysis methods may employ surface meshes extracted in real-time without degenerate triangles.

Marching Cubes (MC) was exhaustingly exploited throughout the thesis to provide the reader familiarity with isosurface polygonizers. The algorithm is well accepted for its robustness, efficiency and simplicity. However, MC has some quality shortcomings, which were presented along some solutions in the literature. MC may extract smooth and badly shaped surface meshes, which implicate in problems enumerated in the thesis.

Our first and second goals, high-quality and efficient polygonization of isosurfaces, were achieved with the mapping of Dual Contouring (DC) and Macet to the GPU, respectively. Macet improved the shape of triangles in MC on-the-fly, by the computation of an extra module in GPU-based MC versions from the literature. Dual Contouring approach was not so straightforward to implement in a GPU-based version. We separated it in two steps, turning it similar to MC.

The first step of our novel DC approach is table-driven. It uses a table for the generation of geometry and specification of topology, which is simple and robust for parallel implementations and which works for other dual algorithms as well. The second step is the approximation of the isosurface. We replaced the minimization of the quadratic error function of DC for a particle-based fashion. The particle-based approach is a straightforward implementation and is not computationally expensive as SV or QR Decompositions. The implementation of both table-driven and particle-based steps together had little overhead in comparison to the most efficient version of MC.

We analyzed common isosurface extractors and pointed advantages and disadvantages of their use. Both DC and Macet approaches have different quality purposes, so each one has its own advantages and weaknesses. If a user needs a surface with better representation of small features, he might choose the polygonization of Dual Contouring. On the other hand, if he needs to use the mesh in some numerical simulation, we presented Macet. Our main focuses were both quality and speed, but other important surface properties such as manifold and watertight were shown on detail as well.

This work also discussed different GPU implementations of isosurface polygonizers and technologies involved. We discussed how to explore the parallel nature of graphics processors, while avoiding some of its constraints. We showed that our approaches for Macet and DC were almost as fast as current MC GPU implementations, while achieving higher quality.

The hardware profiling in the last chapter showed the multiple variables involved in the polygonization of isosurfaces on the GPU. We concluded that it is interesting to use Histogram Pyramids (HP) data structure instead of geometry shader (GS), even with the memory demand and multiple shader state changes of the data structure. GS is a very powerful shading module, but its peak performance is compromised by the number of output scalars required for MC and DC. The profiling also indicated that our memory management for large datasets left the graphics hardware idle with HP, because the GPU was not fed with the multiple partitions fast enough.

One of the objectives of the original Dual Contouring, the mesh adaptively refined by means of octree subdivision, was not explored by our work. It is used for simplification on the mesh regions with low curvature, which is an important tool to store large meshes. The use of our particle-based approach instead of QR (or SV) decomposition does not allow the quad mesh simplification with a simple sum of QEFs.

The next generation of GPUs technology, the Directx11, was presented in a Siggraph conference (BOYD, 2008). It has three new stages in the graphics pipeline: the hull shader, the tesselator and the domain tesselator. A deep analysis of these steps might improve even further the interactivity in the polygonization of isosurfaces. In addition, our HP approaches used pyramids with memory sizes specified by the volume dimensions. But what if we used a static size for the pyramid base, small enough for the inclusion of the volume and other structures? This might be a more efficient approach for HP. Moreover, we could use GS in multiple passes for the creation of geometry, not exceeding the 20 scalars advised by the programming guide of nVidia. This might be a solution for the efficiency in GS.

REFERENCES

BAINES, L. **A teacher's guide to multisensory learning** : improving literacy by engaging the senses. Alexandria, Va.: Association for Supervision and Curriculum Development, 2008. 208p. Portion of title: Multisensory learningLB1576 .B245 2008 1 NEW-BOOKS.

BAJAJ, C. L.; PASCUCCI, V.; SCHIKORE, D. R. Fast Isocontouring for Improved Interactivity. In: VVS '96: PROCEEDINGS OF THE 1996 SYMPOSIUM ON VOLUME VISUALIZATION, 1996, Piscataway, NJ, USA. **...** IEEE Press, 1996. p.39–ff.

BLOOMENTHAL, J. Polygonization of Implicit Surfaces. **Computer Aided Geometric Design**, [S.l.], v.5, p.341–355, 1988.

BLYTHE, D. The Direct3D 10 system. **ACM Trans. Graph.**, New York, NY, USA, v.25, n.3, p.724–734, 2006.

BOYD, C. **The DirectX 11 Compute Shader**. 2008.

CHEN, L. S. et al. Surface Shading in the Cuberille Environment. **Computers Graphics and Applications, IEEE**, [S.l.], v.5, p.33–43, 1985.

CIGNONI, P. et al. Speeding Up Isosurface Extraction Using Interval Trees. **IEEE Transactions on Visualization and Computer Graphics**, Piscataway, NJ, USA, v.3, n.2, p.158–170, 1997.

CIGNONI, P.; ROCCHINI, C.; SCOPIGNO, R. Metro: measuring error on simplified surfaces. **Computer Graphics Forum**, [S.l.], v.17, n.2, p.167–174, 1998.

CO, C. S.; HAMANN, B.; JOY, K. I. Iso-Splatting: a point-based alternative to isosurface visualization. In: PG '03: PROCEEDINGS OF THE 11TH PACIFIC CONFERENCE ON COMPUTER GRAPHICS AND APPLICATIONS, 2003, Washington, DC, USA. **...** IEEE Computer Society, 2003. p.325.

DEY, T. K.; GOSWAMI, S. Tight cocone: a water-tight surface reconstructor. In: SM '03: PROCEEDINGS OF THE EIGHTH ACM SYMPOSIUM ON SOLID MODELING AND APPLICATIONS, 2003, New York, NY, USA. **...** ACM, 2003. p.127–134.

DEY, T. K.; LEVINE, J. A. Delaunay Meshing of Isosurfaces. In: SMI '07: PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON SHAPE MODELING AND APPLICATIONS 2007, 2007, Washington, DC, USA. **...** IEEE Computer Society, 2007. p.241–250.

DIETRICH, C. A. et al. Edge Transformations for Improving Mesh Quality of Marching Cubes. **IEEE Transactions on Visualization and Computer Graphics**, [S.l.], 2009.

DIETRICH, C. A. et al. Marching Cubes without Skinny Triangles. **Computing in Science and Engineering**, Los Alamitos, CA, USA, v.11, n.2, p.82–87, 2009.

DIETRICH, C. et al. Edge Groups: an approach to understanding the mesh quality of marching methods. **14(6)**, [S.l.], v.IEEE Transactions on Visualization and Computer Graphics, p.1651–1658, 2008.

DYKEN, C. et al. High-speed Marching Cubes using HistoPyramids. In: COMPUTER GRAPHICS FORUM, 2008. **...** The Eurographics Association and Blackwell Publishing, 2008. v.27, n.8, p.2028–2039.

EISEMANN, E.; DéCORET, X. Fast scene voxelization and applications. In: I3D '06: PROCEEDINGS OF THE 2006 SYMPOSIUM ON INTERACTIVE 3D GRAPHICS AND GAMES, 2006, New York, NY, USA. **...** ACM, 2006. p.71–78.

EYLES, J. et al. PixelFlow: the realization. In: HWWS '97: PROCEEDINGS OF THE ACM SIGGRAPH/EUROGRAPHICS WORKSHOP ON GRAPHICS HARDWARE, 1997, New York, NY, USA. **...** ACM, 1997. p.57–68.

GIBSON, S. F. F. Constrained Elastic Surface Nets: generating smooth surfaces from binary segmented data. In: MEDICAL IMAGE COMPUTING AND COMPUTER ASSISTED INTERVENTATION MICCAI'98, 1998. **...** Springer Berlin / Heidelberg, 1998. v.1496, p.888.

GOETZ, F.; JUNKLEWITZ, T.; DOMIK, G. Real-Time Marching Cubes on the Vertex Shader. **Eurographics 2005 Short Presentations**, [S.l.], August 2005.

GOURAUD, H. Continuous shading of curved surfaces. In: **Seminal graphics**: poineering efforts that shaped the field. New York, NY, USA: ACM, 1998. p.87–93.

GREGORSKI, B. et al. Interactive view-dependent rendering of large isosurfaces. In: VIS '02: PROCEEDINGS OF THE CONFERENCE ON VISUALIZATION '02, 2002, Washington, DC, USA. **...** IEEE Computer Society, 2002. p.475–484.

GUIBAS, L. J.; STOLFI, J. Primitives for the Manipulations of General Subdivisions and the Computation of Voronoi Diagrams. **ACM Transactions on Graphics**, [S.l.], v.4, n.2, p.74–123, Apr. 1985.

HALL, M.; WARREN, J. Adaptive Polygonalization of Implicitly Defined Surfaces. **IEEE Comput. Graph. Appl.**, Los Alamitos, CA, USA, v.10, n.6, p.33–42, 1990.

HARRIS, M. J. et al. Simulation of cloud dynamics on graphics hardware. In: HWWS '03: PROCEEDINGS OF THE ACM SIGGRAPH/EUROGRAPHICS CONFERENCE ON GRAPHICS HARDWARE, 2003, Aire-la-Ville, Switzerland, Switzerland. **...** Eurographics Association, 2003. p.92–101.

HARRIS, M.; SENGUPTA, S.; OWENS, J. D. Parallel Prefix Sum (Scan) with CUDA. In: NGUYEN, H. (Ed.). **GPU Gems 3**. [S.l.]: Addison Wesley, 2007.

HERMAN, G. T.; UDUPA, J. K. Display of 3D Digital Images: computational foundations and medical applications. In: MEDCOMP '82, 1982. **...** IEEE Computer Society Press, 1982. p.308–314.

HORN, D. Stream Reduction Operations for GPGPU Applications. In: PHARR, M. (Ed.). **GPU Gems 2**: programming techniques for high performance graphics and general purpose computation. [S.l.]: Addison Wesley, 2005. p.573–589.

JOHANSSON, G.; CARR, H. Accelerating marching cubes with graphics hardware. In: CASCON '06: PROCEEDINGS OF THE 2006 CONFERENCE OF THE CENTER FOR ADVANCED STUDIES ON COLLABORATIVE RESEARCH, 2006, New York, NY, USA. **...** ACM, 2006. p.39.

JU, T. et al. Dual contouring of hermite data. **ACM Trans. Graph.**, New York, NY, USA, v.21, n.3, p.339–346, 2002.

KAUFMAN, A. Efficient algorithms for 3D scan-conversion of parametric curves, surfaces, and volumes. In: SIGGRAPH '87: PROCEEDINGS OF THE 14TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, 1987, New York, NY, USA. **...** ACM, 1987. p.171–179.

KAUFMAN, A.; SHIMONY, E. 3D scan-conversion algorithms for voxel-based graphics. In: SI3D '86: PROCEEDINGS OF THE 1986 WORKSHOP ON INTERACTIVE 3D GRAPHICS, 1987, New York, NY, USA. **...** ACM, 1987. p.45–75.

KIPFER, P.; WESTERMANN, R. GPU Construction and Transparent Rendering of Iso-Surfaces. In: VISION, MODELING AND VISUALIZATION 2005, 2005. **Proceedings...** IOS Press: infix, 2005. p.241–248.

KLEIN, T.; STEGMAIER, S.; ERTL, T. Hardware-accelerated reconstruction of polygonal isosurface representations on unstructured grids. **Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on**, [S.l.], p.186–195, Oct. 2004.

KOBBELT, L. P. et al. Feature sensitive surface extraction from volume data. In: SIGGRAPH '01: PROCEEDINGS OF THE 28TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, 2001, New York, NY, USA. **...** ACM Press, 2001. p.57–66.

KRUGER, J.; WESTERMANN, R. Acceleration Techniques for GPU-based Volume Rendering. In: VIS '03: PROCEEDINGS OF THE 14TH IEEE VISUALIZATION 2003 (VIS'03), 2003, Washington, DC, USA. **...** IEEE Computer Society, 2003. p.38.

LEWINER, T. et al. Efficient Implementation of Marching Cubes' Cases with Topological Guarantees. **Journal of Graphics Tools**, [S.l.], v.8, n.2, p.1–15, 2003.

LIVNAT, Y.; SHEN, H.-W.; JOHNSON, C. R. A Near Optimal Isosurface Extraction Algorithm Using the Span Space. **IEEE Transactions on Visualization and Computer Graphics**, Piscataway, NJ, USA, v.2, n.1, p.73–84, 1996.

LOPES, A.; BRODLIE, K. Improving the Robustness and Accuracy of the Marching Cubes Algorithm for Isosurfacing. **IEEE Transactions on Visualization and Computer Graphics**, Piscataway, NJ, USA, v.9, n.1, p.16–29, 2003.

LORENSEN, W. E.; CLINE, H. E. Marching cubes: a high resolution 3d surface construction algorithm. In: SIGGRAPH '87: PROCEEDINGS OF THE 14TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, 1987, New York, NY, USA. **...** ACM Press, 1987. p.163–169.

MACEDONIA, M. R. The GPU Enters Computing's Mainstream. **IEEE Computer**, [S.l.], v.36, n.10, p.106–108, 2003.

NIELSON, G. M. On Marching Cubes. **IEEE Transactions on Visualization and Computer Graphics**, [S.l.], v.9, n.3, p.283–297, 2003.

NIELSON, G. M.; HAMANN, B. The asymptotic decider: resolving the ambiguity in marching cubes. In: VIS '91: PROCEEDINGS OF THE 2ND CONFERENCE ON VISUALIZATION '91, 1991, Los Alamitos, CA, USA. **...** IEEE Computer Society Press, 1991. p.83–91.

NING, P.; BLOOMENTHAL, J. An evaluation of implicit surface tilers. **Computer Graphics and Applications, IEEE**, [S.l.], v.13, n.6, p.33–41, Nov 1993.

NVIDIA. **NVIDIA nfiniteFX Engine**: programmable pixel shaders. [S.l.]: nVidia Corporation, 2003.

NVIDIA. **NVIDIA nfiniteFX Engine**: programmable vertex shaders. [S.l.]: nVidia Corporation, 2003.

NVIDIA. **NVIDIA Performance Toolkit - User Guide**. 2008.

NVIDIA. **GPU Programming Guide GeForce 8 and 9 Series**. 2008.

PASCUCCI, V. Isosurface Computation Made Simple: hardware acceleration, adaptive refinement and tetrahedral stripping. In: IEEE TVCG SYMPOSIUM ON VISUALIZATION (VISSYM), 2004. **...** [S.l.: s.n.], 2004. p.293–300.

PÉBAY, P. P.; BAKER, T. J. **Analysis of triangle quality measures**. 2003. 1817-1839p. v.72, n.244.

RAMAN, S.; WENGER, R. Quality Isosurface Mesh Generation Using an Extended Marching Cubes Lookup Table. In: EUROVIS 2008, 2008. **Proceedings...** [S.l.: s.n.], 2008.

RECK, F. et al. **Realtime isosurface extraction with graphics hardware**. 2004. 33-36p.

SCHAEFER, S.; JU, T.; WARREN, J. Manifold Dual Contouring. **IEEE Transactions on Visualization and Computer Graphics**, Piscataway, NJ, USA, v.13, n.3, p.610–619, 2007.

SCHAEFER, S.; WARREN, J. **Dual contouring**: the secret sauce. [S.l.]: Department of Computer Science, Rice University, 2002. (02-408).

SHEN, H.-W. et al. Isosurfacing in span space with utmost efficiency (ISSUE). In: VIS '96: PROCEEDINGS OF THE 7TH CONFERENCE ON VISUALIZATION '96, 1996, Los Alamitos, CA, USA. **...** IEEE Computer Society Press, 1996. p.287–ff.

SHEWCHUK, J. R. What Is a Good Linear Finite Element? - Interpolation, Conditioning, Anisotropy, and Quality Measures. Unpublished Preprint. **Eleventh International Meshing Roundtable**, [S.l.], p.115–126, 2002.

SNOEYINK, J. Artifacts Caused by Simplicial Subdivision. **IEEE Transactions on Visualization and Computer Graphics**, Piscataway, NJ, USA, v.12, n.2, p.231–242, 2006. Member-Carr,, Hamish and Member-Moller,, Torsten.

TATARCHUK, N.; SHOPF, J.; DECORO, C. Real-Time Isosurface Extraction Using the GPU Programmable Geometry Pipeline. In: SIGGRAPH '07: ACM SIGGRAPH 2007 COURSES, 2007, New York, NY, USA. **...** ACM, 2007. p.122–137.

TREECE, G. M.; PRAGER, R. W.; GEE, A. H. Regularised marching tetrahedra: improved iso-surface extraction. **Computers & Graphics**, [S.l.], v.23, p.583–598, 1999.

VOLVIS. **Datasets Internet source**. 2008.

WILHELMS, J.; GELDER, A. V. Topological considerations in isosurface generation extended abstract. In: VVS '90: PROCEEDINGS OF THE 1990 WORKSHOP ON VOLUME VISUALIZATION, 1990, New York, NY, USA. **...** ACM Press, 1990. p.79–86.

WUENSCHE, B. A Survey and Analysis of Common Polygonization Methods & Optimization Techniques. In: **Machine Graphics & Vision**. [S.l.: s.n.], 1997. v.6, n.4, p.451–486.

WYVILL, G.; MCPHEETERS, C.; WYVILL, B. Data structure for *oft* objects. **The Visual Computer**, [S.l.], v.2, n.4, p.227–234, 1986.

APPENDIX A RESUMO EM PORTUGUÊS

Resumo da Dissertação em Português

A.1 Introdução

A visualização volumétrica é uma área multidisciplinar. Ela envolve desde a Física até a Medicina, campos científicos nos quais é comum a aquisição de grandes volumes de dados. Eles podem ser provenientes, por exemplo, de simulações e ressonâncias magnéticas. Como a quantidade de dados é expressiva, é necessário algum método que filtre uma parte significativa do seu conteúdo. Nosso trabalho utiliza as isosuperfícies para esse fim, que são superfícies implícitas definidas por apenas um parâmetro, o isovalor $I(i) = \{x|f(x) - i = 0\}$. Essas isosuperfícies podem representadas através de polígonos, que são o resultado de técnicas de divisão e conquista como o Marching Cubes (MC) (LORENSEN; CLINE, 1987). A vantagem de se possuir uma representação explícita de conectividade é a possibilidade de usar a informação de topologia em algoritmos de deformação ou ainda em simulações numéricas.

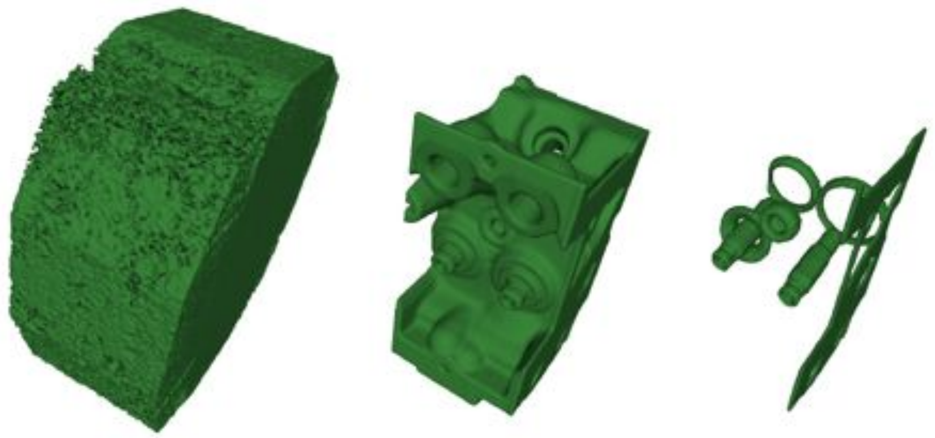


Figura A.1: Navegação entre isosuperfícies. O volume de dados mostra uma tomografia de um motor e as suas múltiplas camadas são apresentadas através de uma navegação interativa.

O nosso primeiro objetivo com esse trabalho é representar essas isosuperfícies com qualidade. Qualidade, no contexto de simulações numéricas (tetraedralizações ou métodos de elementos finitos), se refere a forma desses polígonos. Formas degeneradas

(SHEWCHUK, 2002) podem comprometer os resultados desse tipo de aplicação. MC usualmente gera esse tipo de malha com polígonos degenerados. Por outro lado, qualidade, no contexto de visualização, se refere a aproximação dessas poligonizações das isosuperfícies. Mais especificamente, nos referimos à extração de características acentuadas na isosuperfície, como o Extended MC (KOBBELT et al., 2001) e o Dual Contouring (JU et al., 2002) provêm. Nós buscamos ambas as qualidades neste trabalho: forma dos polígonos e aproximação da isosuperfície.

O nosso segundo objetivo é manter a interatividade na extração dessas isosuperfícies. Isso possibilita com que o cientista navegue através de múltiplos isovalores. Essa navegação facilita a interpretação do conjunto que esses dados representam (Figura A.1). Para este fim, nós buscamos técnicas de aceleração na poligonização de isosuperfícies, como o uso do hardware gráfico (GPUs). Essas GPUs possuem um *hardware* paralelo, que pode diminuir significativamente o tempo de extração em algoritmos de divisão e conquista como os poligonizadores de isosuperfícies.

As contribuições do nosso trabalho se referem aos seguintes pontos do contexto da poligonização de isosuperfícies:

1. O mapeamento para a GPU de poligonizadores de alta qualidade.
    (a) Dual Contouring com um novo método de busca de características acentuadas na isosuperfície.
    (b) Dual Contouring orientado a tabela.
    (c) Marching Cubes com qualidade nos triângulos através do Macet (DIETRICH et al., 2009).
2. A poligonização interativa em volumes grandes.
3. A proposta de melhoramentos na arquitetura das GPUs.

A.2 Poligonização na GPU

Marching Cubes (MC) é um algoritmo fundamental para a poligonização de isosuperfícies. O método é amplamente aceito pela comunidade científica, por sua robustez e simplicidade. O algoritmo consiste basicamente em dividir o espaço em uma grade de cubos tão pequenos que a isosuperfície fique previsível dentro de cada dos cubos. Triangulações são pré-estabelecidas em uma tabela para as diferentes configurações de casos. Essa tabela é criada e consultada através da diferença de sinal dos vértices das arestas dos cubos. Se uma amostra de uma dessas arestas está dentro da isosuperfície seu valor será negativo, senão será positivo. MC é tão importante para a visualização quanto é referenciado em modelagem, outra área importante da computação gráfica. Conjuntos de pontos são uma boa maneira para se representar uma superfície implícita, pois operações *booleanas* (CSG) são facilmente implementadas sobre esses conjuntos. MC extrai malhas de triângulos em grandes volumes de dados na GPU em tempos interativos com o uso da tecnologia de *geometry shading* ou através de uma estrutura de dados chamada Histogram Pyramids (DYKEN et al., 2008). Porém, MC possui alguns inconvenientes, como a geração de triângulos com aspecto ruim.

Macet aparece como uma alternativa viável para a implementação na GPU de um algoritmo que melhora a qualidade dos triângulos do MC. Ele é composto por módulos extras na implementação do MC original, o que impacta significativamente na eficiência do algoritmo na CPU. Porém, esse custo é amortizado na implementação paralela da GPU, já que múltiplos processadores estão lidando com diferentes cubos ao mesmo tempo. Macet melhorou em pelo menos mais de 200 vezes a qualidade do pior triângulo da malha do MC e continuou com tempos interativos. Porém, MC e Macet ainda possuem o problema de apenas extrairem bem malhas suaves, sem considerar características acentuadas nas isosuperfícies.

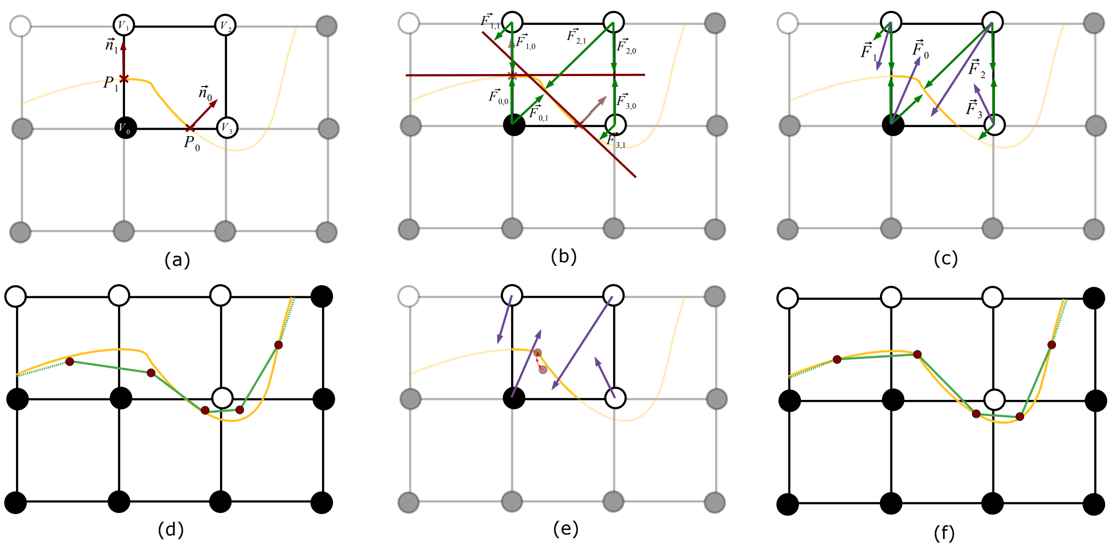


Figura A.2: Novo método para a aproximação de isosuperfícies no Dual Contouring (exemplo em 2D). (a): pontos com suas normais. (b): vetores (setas verdes) dos vértices da grade para os planos definidos pelos pontos com suas normais; (c): forças $\vec{F_k} = \vec{F_i} + \vec{F_{i+1}} + ...$ nos vértices da grade; (d): posicionamento inicial da linha usando o centro de massa $\bar{C}$; (e): movimento da partícula em direção à isosuperfície. As setas vermelhas mostram o caminho que a partícula tomou. (f) poligonização final.

Dual Contouring resolve o problema da suavização das malhas. O método aproxima melhor as malhas não suaves através das normais contidas sobre os pontos da triangulação do MC. Se duas normais nos vértices de uma malha extraída pelo MC possuem um ângulo significativo entre elas (noventa graus, por exemlo), provavelmente ali há uma característica acentuada. Além disso, esse método de poligonização, como o próprio nome já diz, gera malhas duais às do MC. Os pontos dos polígonos do DC encontram-se nas faces do MC. Isso implica numa maior liberdade na movimentação de pontos, já que o MC considera o corte da isosuperfície sobre a aresta da grade, e não dentro do cubo. Nós modificamos a implementação do posicionamento desses vértices duais para uma versão robusta e simplificada do DC, como mostra a Figura A.2, o que acelera o mapeamento para a GPU. Também implementamos um DC orientado a tabela, da mesma maneira do MC, para aumentar ainda mais a eficiência de sua implementação paralela. Os resultados disso foram um novo DC que extrai malhas de qualidade em tempos interativos como o MC (resultados visuais são apresentados na Figura A.3).

Finalmente, descobrimos como a GPU poderia ser melhorada no contexto de poligonização de isosuperfícies. O *geometry shader* (GS) impacta significativamente na performance da GPU, mesmo com a saída de um número pequeno de primitivas por célula do MC e DC. Então optamos pelo *Histogram Pyramids* (HP) como forma mais eficiente de

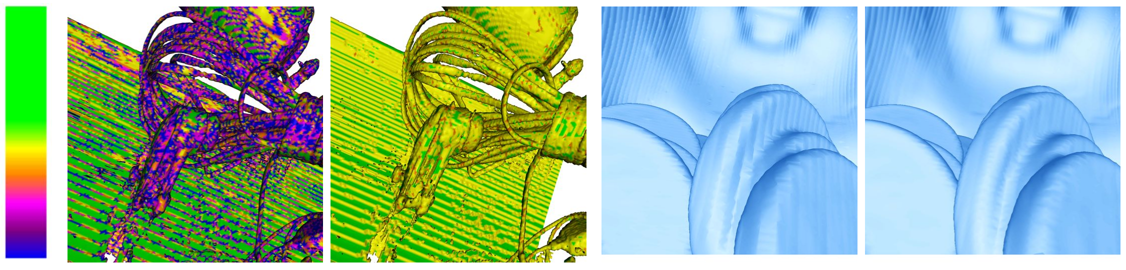

(a) (b) (c) (d)

Figura A.3: Marching Cubes na GPU em (a) e (c) bem como suas versões de alta qualidade com o Macet (b) e com o Dual Contouring (d). O volume de dados *backpack* foi usado em (a) e (b), enquanto que o dataset *pig* foi usado em (c) e (d). A qualidade dos triângulos foi medida usando a razão do raio (*radii-ratio*) em (a) e (b), normalizada de 0 a 1, e as cores foram codificadas usando a tabela ao lado (verde = bom, azul = ruim). Menos artefatos e uma melhor aproximação da poligonização são apresentados em (d) quando comparados a (c).

criação de geometria, mesmo com ambos problemas de mudança constante de estado e uso excessivo de memória na GPU. Se o módulo de *geometry shading* não fosse tão prejudicial para a performance da GPU, não precisaríamos de alternativas como o HP para aumento de performance.